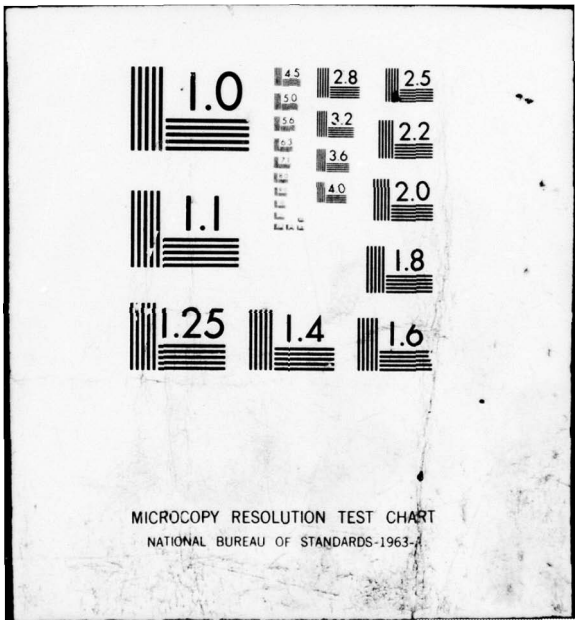
AD-A065 283 STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE F/G 12/1
SET REPRESENTATION AND SET INTERSECTION.(U)
NOV 78 L T PARDO N00014-76-C-0330
UNCLASSIFIED STAN-CS-78-681 NL

| OF |
AD
A065 283

END
DATE
FILMED
4 -79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SET REPRESENTATION AND SET INTERSECTION

by

Luis Trabb Pardo

STAN-CS-78-681
DECEMBER 1978

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Set Representation and Set Intersection

Luis Trabb Pardo

Computer Science Department
Stanford University
Stanford, California 94305

Abstract.

This work discusses the representation and manipulation of sets based on two different concepts: tries, and hashing functions.

The sets considered here are assumed to be static: once created, there will be no further insertions or deletions. For both trie- and hash-based strategies, a series of representations is introduced which together with the availability of preprocessing reduces the average sizes of the sets to nearly optimal values, yet retains the inherently good retrieval characteristics.

The intersection procedure for trie-based representations is based on the traversal in parallel of the tries representing the sets to be intersected, and it behaves like a series of binary searches when the sets to be intersected are of very different sizes. Hashed intersection runs very fast. The average time is proportional to the size of the smallest set to be intersected and is independent of the number of sets (except for the intersection set itself which has to be checked for every set).

This research was supported in part by National Science Foundation grant MCS-77-23738, by Office of Naval Research contract N00014-76-C-0330, and by IBM Corporation. Reproduction in whole or in part is permitted for any purpose of the United States government.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: STAN-CS-78-681
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*: SET REPRESENTATION AND SET INTERSECTION,
5. TYPE OF REPORT & PERIOD COVERED: Technical, November 1978
6. PERFORMING ORG. REPORT NUMBER: STAN-CS-78-681

7. AUTHOR(s): Luis Trabb Pardo
8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0330

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Computer Science Department
Stanford University
Stanford, California 94305

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS:
Office of Naval Research
Department of the Navy
Arlington, Va. 22217

12. REPORT DATE: November 1978
13. NUMBER OF PAGES: 12 [illegible]

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*:
ONR Representative: Philip Surra
Durand Aeromautics Bldg., Rm. 165
Stanford University
Stanford, Ca. 94305

15. SECURITY CLASS. *(of this report)*: Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*:
Releasable without limitations on dissemination.
Technical Rept.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*:
(see reverse side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 094 120 RLH

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

This work discusses the representation and manipulation of sets based on two different concepts: tries, and hashing functions.

The sets considered here are assumed to be static: once created, there will be no further insertions or deletions. For both trie- and hash-based strategies, a series of representations is introduced which together with the availability of preprocessing reduces the average sizes of the sets to nearly optimal values, yet retains the inherently good retrieval characteristics.

The intersection procedure for trie-based representations is based on the traversal in parallel of the tries representing the sets to be intersected, and it behaves like a series of binary searches when the sets to be intersected are of very different sizes. Hashed intersection runs very fast. The average time is proportional to the size of the smallest set to be intersected and is independent of the number of sets (except for the intersection set itself which has to be checked for every set).

Acknowledgments

A Mamá, Abuela y Papo.

Five years ago I was very happy about doing my academic work under (and supported by) Don Knuth. Today, after finishing this thesis, I am even happier, but also for other reasons: Don not only taught me about computer science and guided and encouraged me in my research, but he also offered me his friendship and understanding (to the point of letting me go at my own pace ...). I am very deeply indebted to him for all that.

I want to thank Bob Tarjan and Gio Wiederhold for reading this thesis and providing many useful suggestions. Persi Diaconis introduced me to very interesting results in probability theory.

I want to thank Phyllis Winkler for the many times we laughed together. Phyllis nagged the manuscript out of me and then typed it beautifully.

Chris Van Wyk (Wyke?), world renowned proofreader, did a wonderful job on this thesis.

There is a big old house in Palo Alto where I lived for four years: La Cocotera of Cowper. There (and elsewhere) I shared many things with the many people that maybe did not do much for this thesis but did so much for me: Cristina, Elspeth, Elizabeth, Miguelo, Laurent, Myriam, Manolo, Richard, Margie, Memo, Alan, Kanuto, Mary and David, Dan, Chriso, Arnoldo, et al.

And Ellen and Jonathan.

And Barbara and Jacobo.

And Pata.

Table of Contents

1. Introduction.

Probably nothing would be closer to a "computer sciences panacea" than a general purpose, all-efficient representation for sets. Fortunately the problem has not been solved yet so there are lots of topics for research (and more work for computer scientists ...).

And thus this thesis deals with sets: how to represent them (discussing two different representation techniques) and how to perform some basic operations (intersection in particular) on them.

On Sets and Their Representation.

Throughout this work a set P will be a subset of a universe U. Usually that universe will be the set of positive integers that can be represented with s bits: $U_s = [0,2^s)$. But as a limit case, useful for obtaining asymptotic results, an infinite universe of real numbers will also be considered: $U = [0,1)$.

Since most of the results deal with average cases we need to define our random set P of size $|P| = n$:

(i) In the case of the finite universe $U_s = [0,2^s)$ we will assume all the sets of n elements to be equally probable:

$$\text{Prob}[P = \{p_1, p_2, \ldots, p_n\}] = 1 / \binom{2^s}{n} \quad . \tag{1.1}$$

(ii) For the infinite universe $U = [0,1)$, we will consider the elements of P as independent, uniformly distributed real numbers

in U, so that

$$\text{for any interval } [\alpha,\beta) \subseteq U \text{ , and for any } x \epsilon P \text{ ,} \qquad \text{Prob}[x \in [\alpha,\beta)] = \beta-\alpha \text{ .} \tag{1.2}$$

One of the characteristics of a representation that will be investigated is the size of the average set. We will consistently discuss theoretically attainable estimates of the number of bits needed to store the set. Given actual machine idiosyncracies (addressing, bit handling capabilities, etc.) whoever implements the algorithms might be forced to store bit strings within byte or word boundaries, thus increasing the actual sizes.

The sets will be static, in the sense that once created they don't change, or they change so infrequently that updates may be processed as creation of new sets. (This is the case of many big data bases organized by means of inverted files. Updates are usually kept in a "news" list, and batch-processed periodically.) The importance of this point is that it allows for a reasonable amount of preprocessing when creating each set, making possible faster operations on the sets.

On Set Intersection.

Intersection is the set operation we will investigate in most detail, and again we need to define a statistical model in order to obtain average behaviors.

But, why not simply assume that the sets are random as defined above? We cannot, and the reason is that such an assumption would produce a randomly small intersection, and that is not the case. In general, Jane Q. User has some knowledge of the existence and size of the

intersection: for instance, in a library model (where sets are inverted files of books with a certain attribute) it is very rare that a user will try to retrieve books under both the keywords "Russian literature" and "Ostriches". Thus somehow the intersection size has to be thrown into the equation.

Assume that we want to intersect the sets $P_1, P_2, \ldots, P_m$. Let $\bar{P} = \langle |P_1|, |P_2|, \ldots, |P_m| \rangle$ be the sequence of sizes and $Q = P_1 \cap P_2 \cap \ldots \cap P_m$, $|Q| = q$, the intersection to be computed. We will define the <u>average intersection time</u>, $I(\bar{P}, q)$, according to the following model:

- the intersection set, Q, is any arbitrary set of size q (with the obvious restriction that $q \leq \min\{|P_1|, |P_2|, \ldots, |P_m|\}$),
- the subsets $W_i = P_i - Q$ are random subsets of $U - Q$ in the sense that for all $x \epsilon U$

$$\text{Prob}[x \in W_i] = \frac{|W_i|}{|U - Q|} \quad . \tag{1.3}$$

Notice that given the above definition there is a chance that the W_i's do actually yield a nonempty intersection. But that will only mean that we are getting a larger intersection for the price of a smaller one.

Given the above model, the average intersection time can be divided into

- $V(\bar{P}, q)$, the time needed to verify that Q is a true subset of all P_i's, plus
- $t(\bar{P}, q)$, the time needed (or "wasted") to verify that the intersection of the W_i's is empty.

Thus

$$I(\bar{P}, q) = V(\bar{P}, q) + t(\bar{P}, q) \quad . \tag{1.4}$$

The above discussion does not imply that intersections will be computed in two distinct steps, on the contrary, both algorithms introduced in this work proceed without any knowledge of what Q is and what belongs in the W_i's. But Equation (1.4) underlies the methodology used to obtain bounds for the intersection times.

On the other hand, $V(\bar{P}, q)$ may be seen as a measure of the "productive time": a user will usually be glad to know that s/he has to pay $V(\bar{P}, q)/q$ per element found (in the intersection). Similarly $t(\bar{P}, q)$ is a measure of the "risk" involved in computing the intersection and finding nothing.

On Set Union.

In certain representations, computing the union of two sets requires essentially the same time as computing their intersection. Though this work will not present any detailed account of algorithms to compute unions, a short discussion, mainly for purposes of comparison with current results, will be included in the last chapter.

A Gourmet Guide to the Contents.

Two representations of sets are discussed in this thesis. The first one, based on tries, is covered in Chapters 2 and 3. Chapter 2 discusses issues about trie representation (this chapter relies heavily on Appendix A, which presents some needed counting results about tries) and Chapter 3 discusses the trie intersection algorithm.

The second representation, based on hashing techniques, is presented in Chapter 4. Again the representation proper is discussed first (Section 4.1) after which come the intersection methods (Section 4.2). Finally, Chapter 5 presents the conclusions and practical considerations.

It is traditional for the first chapter of a thesis to state how much of the material is "new" as opposed to well-known. In this case, essentially all of the algorithms and analyses to be discussed appear to be new, except for some auxiliary formulas that are quoted from the literature. The basic ideas of hashing and trie representation are, of course, well known, and the analysis of these basic methods are stated in order to demonstrate the improvements being made.

Notation and Background.

The notation used throughout this thesis is pretty standard (so it will not be discussed further) with the exception of some sort of "vector-like" notation required in Chapter 3 (where it is duly presented).

Both the concepts of tries and hash functions are excellently covered by Knuth in [Knuth 75] (Section 6.3 covers tries and Section 6.4 hashing) and the interested reader is referred to it for further details about the above data structures.

2. Trie Representation of Sets.

Knowing the bounds of the universe (and a total order relation over its elements) is the key to representing any subset of such a universe as a trie, and this chapter discusses different ways of doing that.

Throughout this chapter we will be concerned with representations and their space requirements, leaving the efficiency of intersection to Chapter 3.

When talking about space requirements, a measure of the minimum space needed to represent a set is always handy. Let the universe be

$$U_s = [0, 2^s)$$

and $M(s,n)$ the minimum average number of bits required to represent a subset $S \subseteq U_s$ of $|S| = n$ elements.

Since there are $\binom{2^s}{n}$ subsets of size n it is obvious that

$$M(s,n) \geq \left\lceil \lg \binom{2^s}{n} \right\rceil \geq \lg \binom{2^s}{n} \quad . \tag{2.0.1}$$

And assuming relatively small subsets ($n \leq 2^s/s$ will suffice) we obtain

$$M(s,n) \geq n\left(s - \lg n + \frac{1}{\ln 2}\right) - \frac{1}{2} \lg n + O(1) \quad . \tag{2.0.2}$$

But since $\lg n < s$ we can state that

$$M(s,n) \geq n\left(s - \lg n + \frac{1}{\ln 2}\right) - c \tag{2.0.3}$$

where c is some appropriate constant.

Each different representation presented below will be associated with a subscript and for each one some of the following average values will be investigated:

$t_i(s,n)$: size of tag bits per node,

$b_i(s,n)$: size of a node in bits,

$A_i(s,n)$: number of nodes,

$P_i(s,n)$: total number of bits used to store pointers,

$T_i(s,n)$: total number of bits used to store the set elements,

and $B_i(s,n)$: total size in bits.

(The above quantities refer to "Representation i", and to subsets of n elements from the universe $[0,2^s)$.)

Also if R is a set, the quantities $t_i(s,R)$, $b_i(s,R)$, $P_i(s,R)$, $T_i(s,R)$, $A_i(s,R)$ and $B_i(s,R)$ will denote the actual values for the set R , not averages.

A certain mythical representation that asymptotically realizes the optimum $M(s,n)$ will be our representation 0; thus

$$B_0(s,n) = n\left(s - \lg n + \frac{1}{\ln 2}\right) \quad . \tag{2.0.4}$$

For comparison purposes, the good old sequential allocation (with each cell viewed as a node) will be representation 1 , and

$$\begin{aligned} t_1(s,n) &= 0 & P_1(s,n) &= 0 \\ b_1(s,n) &= s & T_1(s,n) &= ns \\ A_1(s,n) &= n & B_1(s,n) &= ns \quad . \end{aligned} \tag{2.0.5}$$

The following example will be used throughout this chapter.

Example 2.0.1. Let the universe be the set $U_8 = [0,2^8)$; we will use the subset

$$S = \{3 , 25 , 50 , 51 , 130 , 189 , 200 , 232 , 249\} \quad ,$$

or in octal $S = \{003 , 031 , 062 , 063 , 202 , 275 , 310 , 350 , 371\}$. Notice that $|S| = 9$ and the sequential representation of S requires $B_1(8,S) = 8 \times 9 = 72$ bits. The optimum in this case is $B_0(8,9) = 56.5$. □

2.1 Binary Tries.

Bisecting the universe sounds like the most "natural" kind of partition and thus binary tries are born.

The obvious way, representation 2 , uses the data structure depicted in Figure 2.1.1. Under such representations each node requires space for two tags (one bit each) and two fields large enough to hold either a value or a pointer, hence

$$b_2(s,n) = 2(s+1) \quad . \tag{2.1.1}$$

The average number of nodes presents us with some difficulties, since there appears to be no reasonably simple expression for it. The analysis in Appendix A (Section A.1) shows that

$$A_2(s,n) = (2^s-1) - \frac{1}{\binom{2^s}{n}} \sum_{0 \le j < s} \left[2^j \binom{2^s-2^{s-j}}{n} + 2^s \binom{2^s-2^{s-j}}{n-1} \right] , \tag{2.1.2}$$

a really horrible expression.

Fortunately, we can bound $A_2(s,n)$, since in general $A_2(s+1,n) > A_2(s,n)$ (see Appendix A, Lemma A.2.1), and by Lemma A.2.3

$$\lim_{s \to \infty} A_2(s,n) = A(n) \tag{2.1.3}$$

where $A(n)$ is the number of nodes in a trie representation of a random set of n elements in $U = [0,1)$.

In practice the difference between $A_2(s,n)$ and $A(n)$ is very small, since usually the sets to be represented are much smaller than the universe (it is possible to prove that

$$A_2(s,n) = A(n)(1 - O(ns/(2^s-n))) \quad ,$$

see Appendix A, Section A.6, and thus the leading term of $A(n)$ applies to $A_2(s,n)$ when $n = O(2^{s/2})$, the usual case in practical applications).

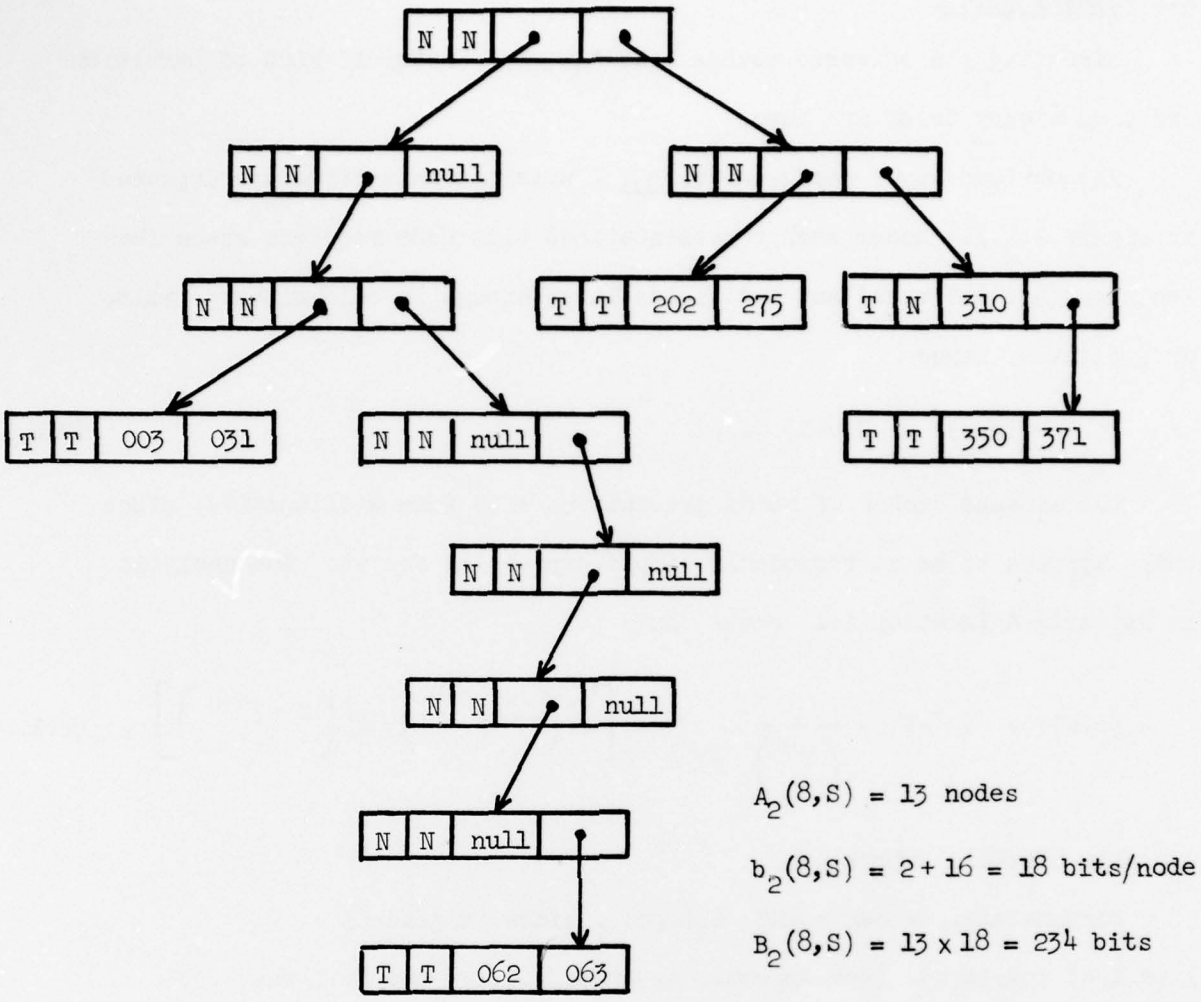


Figure 2.1.1. Representation 2.

Tag fields store values N (nonterminal subtrie) or T (terminal subtrie). The other fields store either a pointer (including the null one) or a set element. A pointer is used when there is more than one element to be represented. The left subtrie at level k represents the subset with bit k equal to 0, the right subtrie represents bit k equal to 1, where bits are numbered 0...7 from left to right. Notice that the set elements appear in octal notation.

For the rest of the chapter we will adopt the estimates for tries in the infinite universe $U = [0,1)$ for the actual case $U_s = [0,2^s)$ whenever the quantities for the infinite universe are upper bounds for the corresponding one in the finite case.

(The interested reader is referred to Section A.3 for further discussion of asymptotic results for the infinite universe case.)

So we will adopt

$$A_2(s,n) = n/\ln 2 \quad , \tag{2.1.4}$$

giving

$$B_2(s,n) = \frac{2}{\ln 2}(s+1)n = n(2.88s + 2.88) \quad . \tag{2.1.5}$$

Space requirements for the above basic representation do not look very good, and they certainly can be improved. A first (and easy) improvement, representation 3, is to eliminate the space wasted in null pointers. (There are $A_2(s,n) - n + 1$ *null pointers requiring* s bits each.) Figure 2.1.2 illustrates representation 3.

Both left and right tags now need to take three different values (terminal, nonterminal, or null subtrie), hence

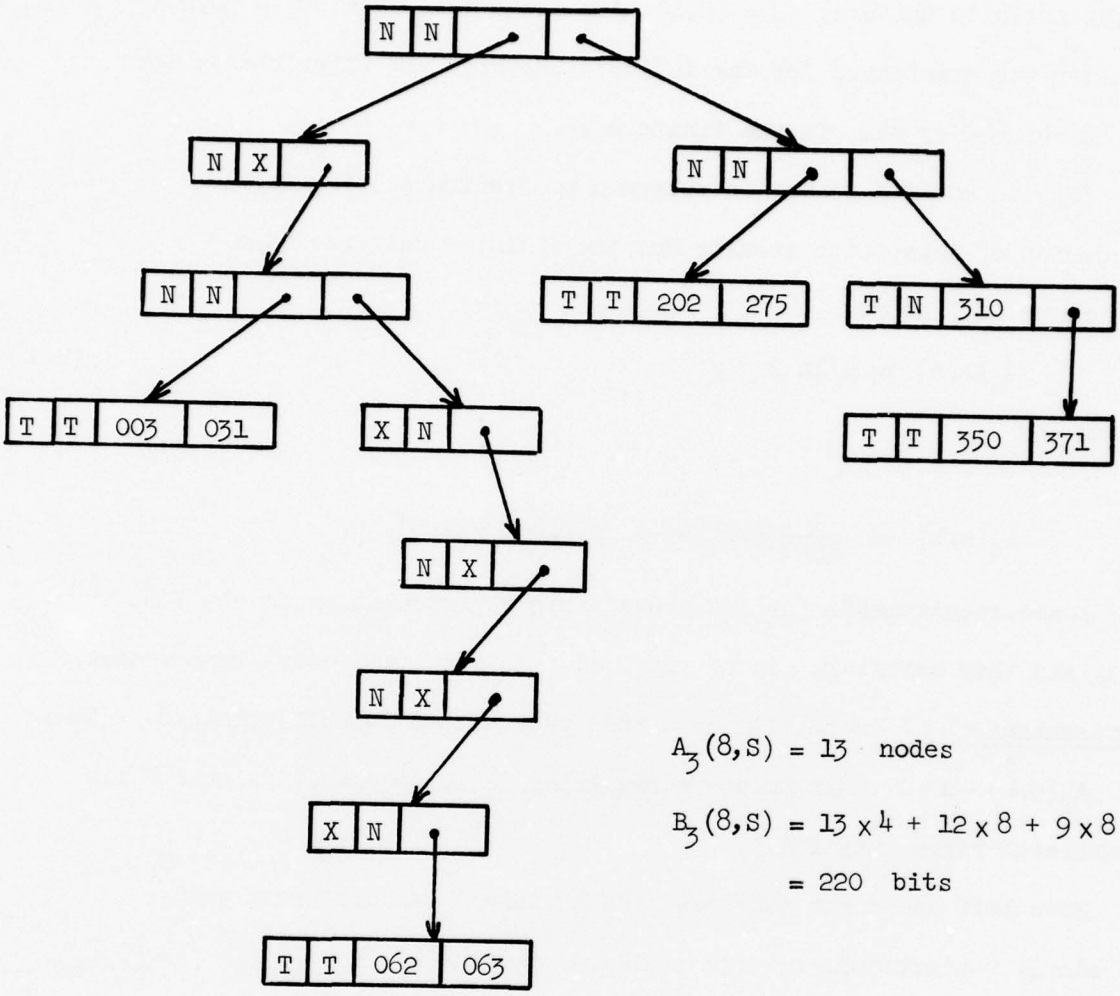
$$t_3(s,n) = 4 \text{ bits} \quad . \tag{2.1.6}$$

The number of nodes is still

$$A_3(s,n) = A_2(s,n) = \frac{n}{\ln 2} \quad . \tag{2.1.7}$$

And since $A_3(s,n)-1$ pointers are needed

$$P_3(s,n) = \frac{n}{\ln 2} s - s \tag{2.1.8}$$



$A_3(8,S)$ = 13 nodes

$B_3(8,S)$ = 13 x 4 + 12 x 8 + 9 x 8

= 220 bits

Figure 2.1.2. Representation 3.

Tags notation

N = nonterminal subtrie

T = terminal subtrie

X = null subtrie.

and

$$T_3(s,n) = n \cdot s \quad . \tag{2.1.9}$$

So

$$B_3(s,n) = 4\frac{n}{\ln 2} + n\left(\frac{1}{\ln 2} + 1\right)s = n(2.44s + 5.77) \quad . \tag{2.1.10}$$

Another way to improve on representation 2 is to store the nodes in some fixed order (Preorder mit Variazionen), thus requiring at most one pointer per node (since the left sibling of a node is always stored right after the node itself). This is representation 4, and is depicted in Figure 2.1.3.

Only six different tag combinations are possible, hence

$$t_4(s,n) = 3 \text{ bits} \quad . \tag{2.1.11}$$

The number of nodes is still the same

$$A_4(s,n) = A_2(s,n) = \frac{n}{\ln 2} \quad . \tag{2.1.12}$$

A pointer is needed only when both subtries are nonterminal. The analysis in Section A.4 shows that about one of every four nodes has two nonterminal siblings, so

$$P_4(s,n) = \frac{1}{4}\,\frac{n}{\ln 2} \cdot s \quad . \tag{2.1.13}$$

In order to store the set elements we need

$$T_4(s,n) = n \cdot s \quad . \tag{2.1.14}$$

Finally,

$$B_4(s,n) = 3\frac{n}{\ln 2} + \frac{1}{4}\,\frac{ns}{\ln 2} + n \cdot s = n(1.36s + 4.33) \quad . \tag{2.1.15}$$

The previous representations dealt with fixed size fields, and not much more can be done by way of improvement. On the other hand, using

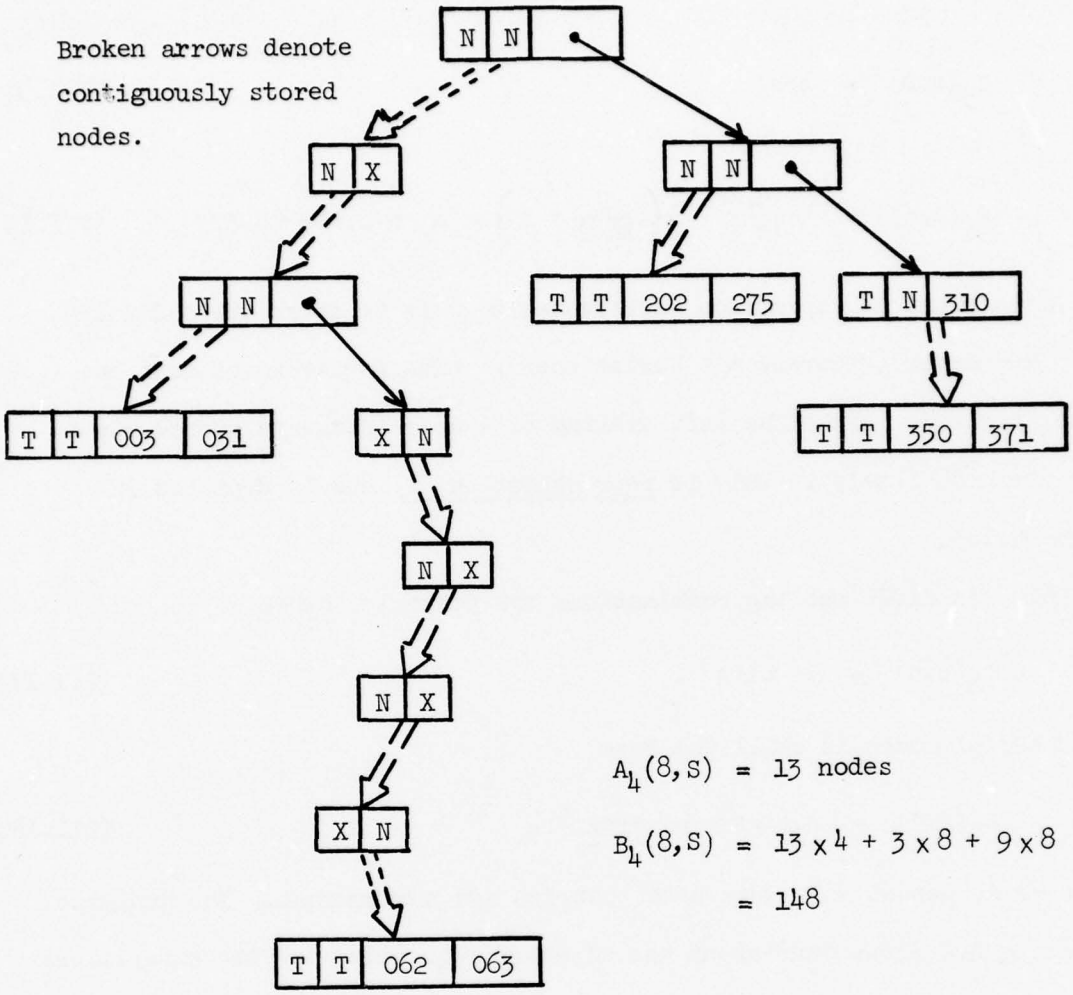


Figure 2.1.3. Representation 4.

Possible tag combinations are NN , NT , TN , TT , NX and XN . Pointer/value field is used as a pointer when tag is NN , and as a value if tag is TN or NT . When the tag is TT the two set elements follow the node. The nodes are stored in preorder except that a terminal right subtrie precedes its nonterminal left brother.

variable length fields we can improve the storage requirements in two ways:

(i) Using the information implicit in the trie structure allows us to store only that suffix of each set element that has not been defined by the path from the root to the parent node.

(ii) Defining pointers as displacements from the current node (and, of course, storing nodes in, say, preorder) requires pointers of size proportional to the logarithm of the number of elements in the subtries.

And so we get to representation 5, similar in organization to representation 4 but with variable length pointer and set element fields (refer to Figure 2.1.4).

Tag size and number of nodes remain the same as before

$$t_5(s,n) = 3$$
$$A_5(s,n) = \frac{n}{\ln 2} \quad . \tag{2.1.16}$$

The space required to store the set elements also needs some thought: we will estimate

$$T_5(s,n) = n \cdot s - n \cdot (\text{average depth}) \tag{2.1.17}$$

where the average depth of a terminal element in the trie has already been discussed by Knuth [Knuth 73, Section 6.3] for the case $s = \infty$. Actually the quantity " $n \cdot$(average depth) " above is the average total number of bit inspections needed to look up the n keys in the set (that is C_n as defined by Equation 6.3(5) in [Knuth 73]), so we obtain

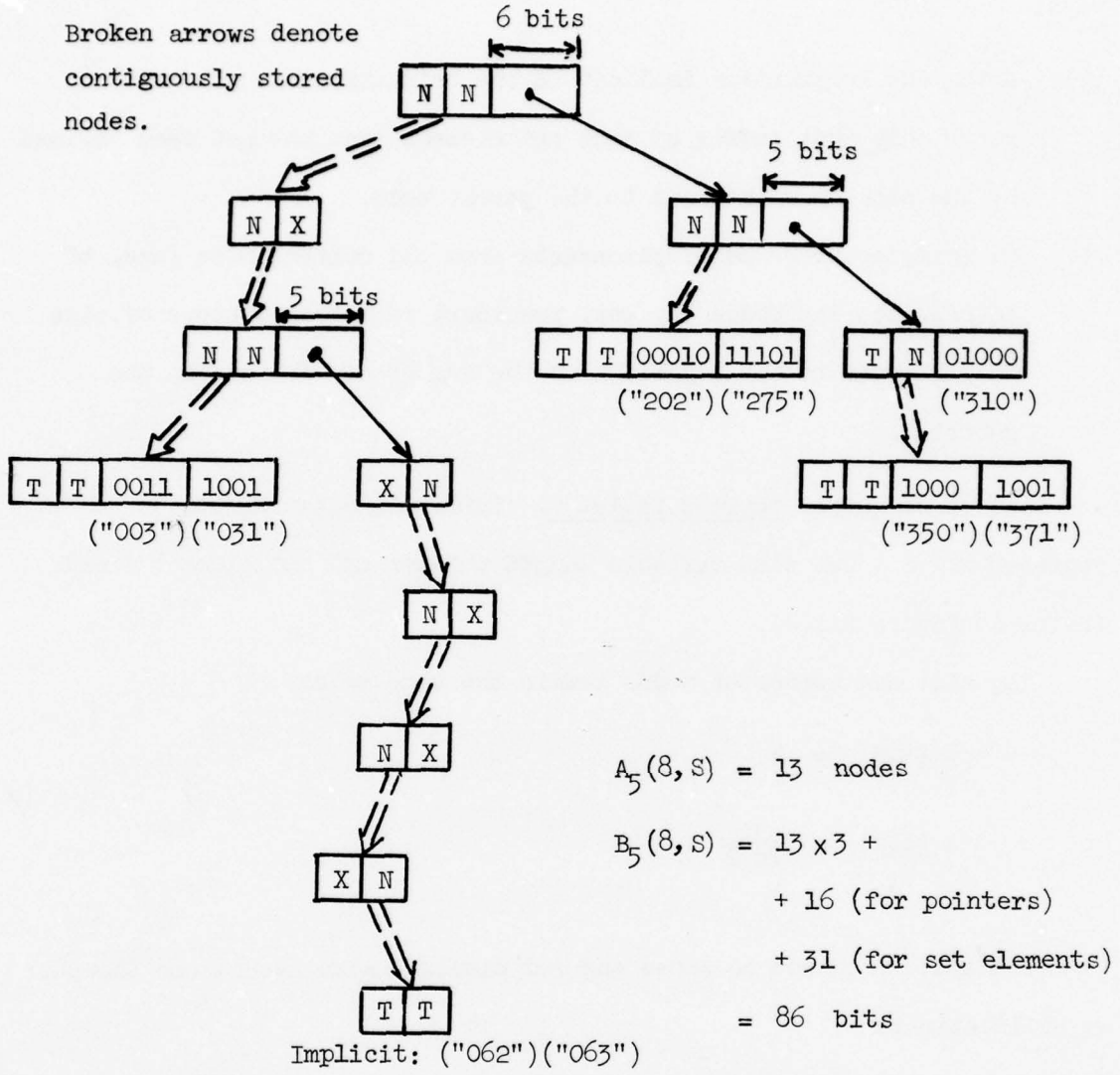


Figure 2.1.4. Representation 5 .

Pointer sizes were assigned arbitrarily, for completeness sake.

(They depend on the pointer encoding scheme used.)

$$T_5(s,n) = n\left[s - \lg n - \frac{\gamma}{\ln 2} - \frac{1}{2} \right] \qquad (2.1.18)$$

A pointer will now measure the displacement (in bits) from the current node to the given (right) subtrie.

Given a node of height d (d is s minus the distance from the root) and with a left subtrie of k elements we will encode the pointer to the right subtrie as a pair $\langle k,\delta \rangle$ so that $(k.d - \delta)$ bits is the size of the left subtrie, and thus the distance to the right one. Notice that the set represented by the left subtrie needs at least $(kd - k \lg k)$ bits on the average, and therefore, also on the average

$$\delta \leq k \lg k$$

so that δ can be encoded with $(1+\varepsilon) \lg k$ bits (where ε is a small positive constant less than $1/(e \cdot \ln 2)$).

So we have discovered that with the above convention, the size of the pointers becomes independent of the size of the universe, so that the root node for a set of n elements requires a pointer of size $(2+\varepsilon) \lg n$. (The term $\varepsilon \lg n$ also absorbs any overhead due to the variable length encoding scheme for the quantities n and δ . See [Even and Rodeh 78] for a possible encoding scheme.)

So now we may compute

$$p_n = (1 - \delta_{n0} - \delta_{n1})(2+\varepsilon) \lg n + 2^{1-n} \sum_{k \geq 2} \binom{n}{k} p_k \qquad (2.1.19)$$

as a good estimate for the total pointer space requirements, $P_5(s,n)$. (Notice that we assign pointer space to every node, even though in practice many nodes don't need it.) Again Appendix A provides us with the answer (Section A.5) and we dutifully adopt it as

$$P_5(s,n) = (2+\varepsilon) \cdot 2.96\, n \qquad (2.1.20)$$

(a remarkable result: about 6 bits per element are needed for pointers).

So allowing ε to increase $P_5(s,n)$ up to, say, $7n$, we obtain

$$B_5(s,n) = \frac{3n}{\ln 2} + 7n + n\ s - \lg n - \frac{\gamma}{\ln 2} - \frac{1}{2}$$

$$= n(2 - \lg n + 10) \quad . \qquad (2.1.21)$$

Tries include chains of one-way branching nodes, whose only function is to convey path information. This clearly suggests the possibility of representing that path as a bit string and getting rid of such nodes, and that turns out to be our representation 6.

In this representation, each node includes a (possibly empty) prefix field, that replaces any eliminated chain immediately preceding the current node, as may be seen in Figure 2.1.5. Representation 6 is somehow related to the idea behind Patricia tries presented in [Morrison 68].

The tag size is

$$t_6(s,n) = 2 \qquad (2.1.22)$$

and the number of nodes

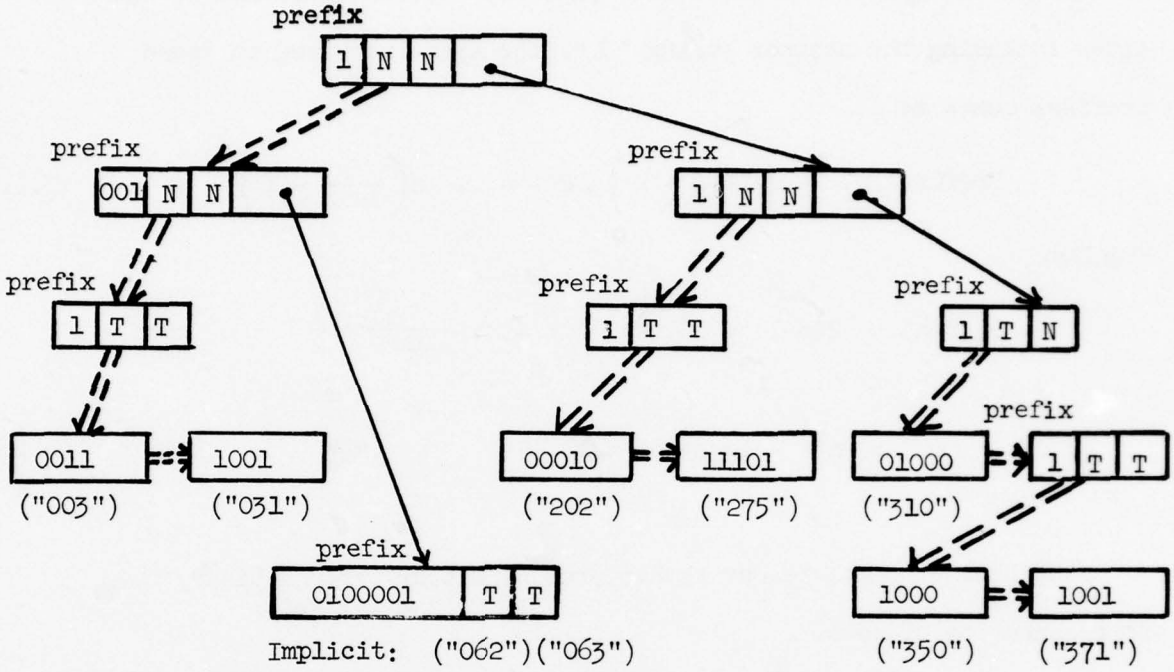
$$A_6(s,n) = n-1 \quad . \qquad (2.1.23)$$

Pointer and set element space are (as in the previous case)

$$P_6(s,n) = P_5(s,n) = (2+\varepsilon) \cdot 2.96\, n \leq 7n$$

$$T_6(s,n) = T_5(s,n) = n\left(s - \lg n - \frac{\gamma}{\ln 2} - \frac{1}{2}\right) \qquad (2.1.24)$$

Broken arrows denote
contiguously stored nodes



$A_6(8,S) = 8$

$B_6(8,S) = 8 \times 2$ (tags) + 16 (prefixes)
+ 16 (pointers)
+ 31 (set elements)
= 79 bits

Figure 2.1.5. Representation 6.

Each node has two descendants, hence possible tag values are N and T.

A prefix $z_1 z_2 \ldots z_p$ is coded as

"$0 z_1 0 z_2 \ldots 0 z_p 1$".

(The empty prefix is simply "1".)

Each one-way branching node in, say, representation 5 generates two bits of prefix space, and there are $[A_5(s,n) - n]$ of those nodes. So after including the stopper string "1", the space required to store prefixes comes to

$$\text{Prefix}(s,n) = \left(\frac{n}{\ln 2} - n\right) \times 2 + n = n\left(\frac{2}{\ln 2} - 1\right) \quad . \qquad (2.1.25)$$

Finally,

$$B_6(s,n) = 2(n-1) + 7n + n\ \left(s - \lg n - \frac{\gamma}{\ln 2} - \frac{1}{2}\right) + n\ \frac{2}{\ln 2} - 1$$

$$= n(s - \lg n + 9.56) \quad . \qquad (2.1.26)$$

And thus we complete our rather prolonged tour through binary trie representation of sets.

A closing remark: it is interesting to notice that it is possible to encode the set <u>and</u> the trie structure and still come within $(c \cdot n)$ bits of the lower space bound (2.0.3).

2.2 Other Representations Based on Tries.

Several variations on the basic binary trie may be used, and this section will simply state some of them without further analysis.

First, there is an obvious generalization of binary to M-ary tries. An M-ary trie representation reduces the average number of nodes to approximately $n/(M \ln 2)$, but it forces an increase in the size of each node. Further, storing the nodes in a fixed order (as in the last three representations of Section 2.1) will only save one out of M pointers. A way to produce more economical M-ary tries is to store only the non-null pointers, though a price is paid in the increase in time needed to process each node for set operations.

An interesting way to represent M-ary tries (compressed tries or C-tries) is presented by Maly [Maly 76]. A C-trie stores only a pointer to the first sibling of each node, and stores all the siblings sequentially.

As an important consequence of using M-ary tries, the parameter M may be chosen in order to minimize the "waste" if the data structure is to be stored within byte or word boundaries.

A second idea follows the old paradigm "keep small things simple...": there is no better way to store a small set than a simple one (e.g., a sequentially stored list). More precisely, store small subsets (those of size less than a certain parameter q) sequentially. This scheme will certainly reduce the overall complexity of the data structure, though it complicates the implementation of some set operations.

A combination of the M-ary trie and the small subset sequential storing ideas is analyzed by Knuth [Knuth 73, Exercise 6.3.20]. That analysis shows that the average number of nodes is reduced to $n/(M \cdot q \cdot \ln 2)$. Further savings are obtained by storing only the suffixes of the elements of each

small set, and an example of such a "combo" technique is presented in Figure 2.2.1.

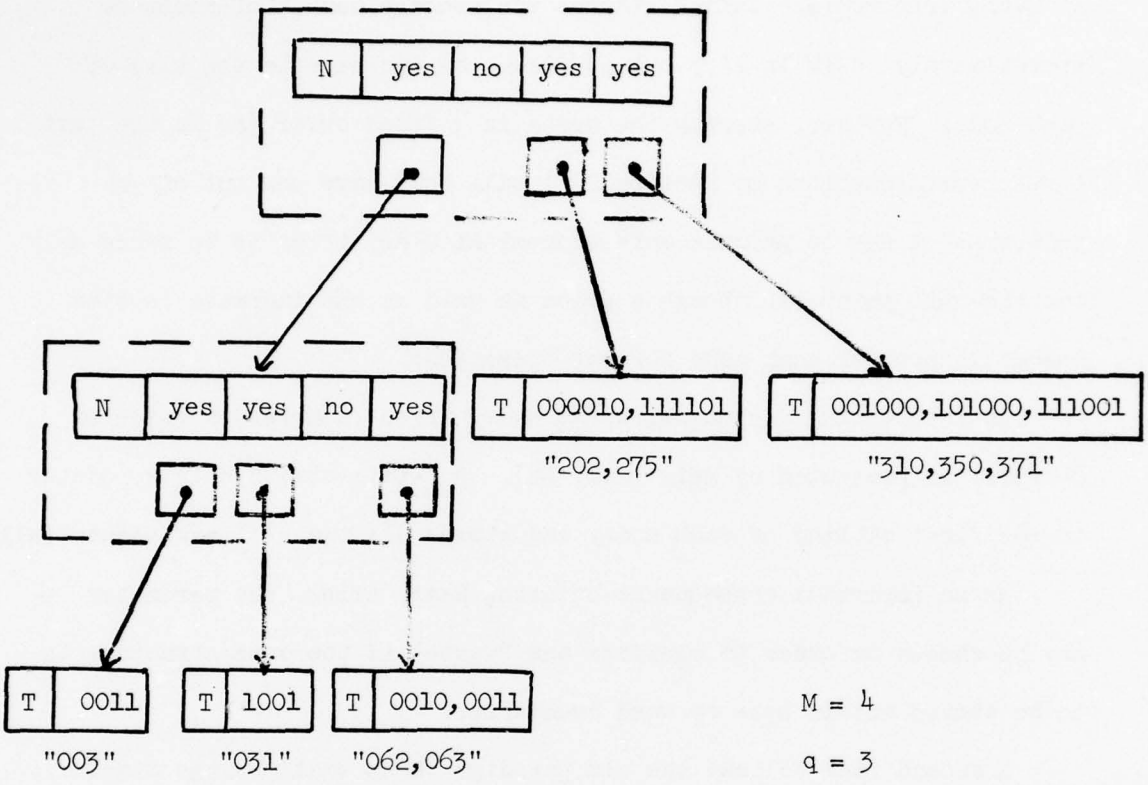


Figure 2.2.1. A "combo" representation, combining an M-ary trie and the representation of sets of size less than q as sequential lists of the suffixes of each element.

3. The Trie Intersection Algorithm.

Once two or more subsets of a given universe U have been represented as tries in a consistent manner, it is possible to take advantage of such representations for the purpose of computing their intersection (or their union, though the union algorithm will be discussed later).

Let us assume two subsets P_1 and P_2 of a universe U; both P_1 and P_2 are represented as tries according to the partition $U = U' \cup U''$ (thus P_1 has subtries $P_1' = P_1 \cap U'$ and $P_1'' = P_1 \cap U''$ and, similarly, $P_2' = P_2 \cap U'$ and $P_2'' = P_2 \cap U''$). Now the intersection $P_1 \cap P_2$ (within the universe U) can be computed as follows:

$$\text{Intersection}(P_1, P_2, U)$$

$$= \text{Intersection}(P_1', P_2', U') \cup \text{Intersection}(P_1'', P_2'', U'') \qquad (3.0.1)$$

but since both intersections on the right hand are disjoint, the union is simply a juxtaposition, requiring constant time.

The above recursion should proceed until the computation becomes trivial, either because both sets are small enough or because one of them is empty. And that is where the savings are: when either one of the sets is empty, the other one can be disregarded immediately. On the average the above situation arises very often, especially if the starting sets are of different sizes.

In order to analyze the method, a measure has to be introduced, and the number of trie nodes visited seems plausible since the amount of work work at each node is bounded by a constant.

So we will now pick up the discussion about intersection time started in the introduction. Again, assume the sets $P_1, P_2, \ldots, P_m$ are represented as binary tries within a universe U, with $\bar{P} = \langle |P_1|, |P_2|, \ldots, |P_m| \rangle$ the

sequence of sizes and Q, $|Q| = q$, their "arbitrary" intersection.

We will estimate the average intersection time (given in Equation (1.4)) as

$$I(\bar{P},q) \leq V(\bar{P},q) + t(\bar{P}) \qquad (3.0.2)$$

where $V(\bar{P},q)$ is as before and $t(\bar{P})$ is a bound on the wasted effort $t(\bar{P},q)$.

In order to bound $t(\bar{P},q)$, notice that since

$$|W_i| = |P_i - Q| \leq |P_i| \quad , \qquad (3.0.3)$$

verifying that the intersection of the random subsets W_i is empty and must take less time than computing the overall intersection of <u>totally random sets</u> of sizes $|P_i|$ (that is, sets whose intersection size is <u>not</u> arbitrarily preset)

$$t(\bar{P},q) \leq I(\bar{P},0) = t(\bar{P}) \quad . \qquad (3.0.4)$$

Thus, Section 3.1 analyzes $t(\bar{P})$ as the running time of the algorithm over random sets in a universe $U_s = [0,2^s)$, and produces a hopelessly inscrutable answer, so Section 3.2 analyzes the limiting case of the universe of reals $U = [0,1)$ that allows for an asymptotic expansion for $t(\bar{P})$ (presented in Section 3.3). Finally, Section 3.4 analyzes the overall intersection time and discusses some examples.

<u>A Bit of Notation</u>.

Most of the equations in this chapter will deal with m-tuples, and a concise notation is needed. An m-tuple will be denoted by a letter with an overbar; let $\bar{A} = \langle a_1, a_2, \ldots, a_m \rangle$ and $\bar{B} = \langle b_1, b_2, \ldots, b_m \rangle$. A scalar v will expand to the m-tuple $\bar{v} = \langle v, v, \ldots, v \rangle$.

Operators will be extended to apply to m-tuples

$$\bar{A} * \bar{B} = \langle a_1 * b_1, \ldots, a_m * b_m \rangle \quad \text{for "} * \text{" denoting basic arithmetic operations}$$

$$\bar{A}^{\bar{B}} = \langle a_1^{b_1}, \ldots, a_m^{b_m} \rangle \qquad \delta_{\bar{A}\bar{B}} = \langle \delta_{a_1 b_1}, \ldots, \delta_{a_m b_m} \rangle$$

$$\bar{A}! = \langle a_1!, \ldots, a_m! \rangle \qquad \binom{\bar{A}}{\bar{B}} = \left\langle \binom{a_1}{b_1}, \ldots, \binom{a_m}{b_m} \right\rangle .$$

Operations on scalars and m-tuples will denote the coercion of the scalar to the corresponding m-tuple followed by the operation itself. For instance:

$$\binom{\alpha}{\bar{A}} = \binom{\bar{\alpha}}{\bar{A}} = \left\langle \binom{\alpha}{a_1}, \ldots, \binom{\alpha}{a_m} \right\rangle .$$

Finally some operators will map m-tuples onto scalars

$$\sum \bar{A} = a_1 + \ldots + a_m \qquad \prod \bar{A} = a_1 \cdot \ldots \cdot a_m$$

$$\bar{A} \leq \bar{B} = (a_1 \leq b_1) \wedge \ldots \wedge (a_m \leq b_m)$$

$$\bar{A} \overset{\leq}{\neq} \bar{B} = (\bar{A} \leq \bar{B}) \wedge ((a_1 \neq b_1) \vee \ldots \vee (a_m \neq b_m)) .$$

3.1 Wasting Effort in the Finite Universe.

Assume that the sets $P_1, P_2, \ldots, P_m$ are random subsets of the universe $U_s = [0, 2^s)$, and have sizes $\bar{P} = \langle p_1, p_2, \ldots, p_m \rangle$ respectively. We are interested in the running time $t_s(\bar{P})$ of the intersection algorithm for random sets $P_1, P_2, \ldots, P_m$.

Given that the binary trie representation partitions the universe into $U' = [0,2^{s-1})$ and $U'' = [2^{s-1},2^s)$, we have that for any sequence of sets $P_1,\ldots,P_i,\ldots,P_m$, such that $|P_i| = p_i$, the algorithm will take the lower half partitions $P_1' = P_1 \cap U', \ldots, P_m' = P_m \cap U'$ and intersect them, and then do the same for the upper half partitions $P_i'' = P_i \cap U''$.

Hence, assuming non-empty sets, $\bar{0} \overset{<}{\neq} \bar{P} \leq \overline{2^s}$, and $s \geq 1$,

$$t_s(\bar{P}) = \sum_{\bar{0} \leq \bar{R} \leq \bar{P}} Part(\bar{P},\bar{R},s)\cdot[t_{s-1}(\bar{R}) + t_{s-1}(\bar{P}-\bar{R})] + v \qquad (3.1.1)$$

where

$Part(\bar{P},\bar{R},s)$: probability of a partition $\bar{R}$ (lower half) and $\bar{P}-\bar{R}$ (upper half) of a sequence of sets of size $\bar{P}$;

v: time required to visit the m roots of the tries representing the sets.

Equation (3.1.1) does not cover the cases where at least one of the sets to be intersected is empty, or the boundary case when the universe is of size 1. So

$$\begin{cases} \text{for } \bar{P}\text{: } \exists i,\ p_i = 0 \text{ let } t_s(\bar{P}) = 0 \\ \qquad\quad \text{(the intersection is known to be empty)} \\ \text{for } s = 0 \quad t_0(\bar{P}) = 0\ . \end{cases} \qquad (3.1.2)$$

Now the probability of a set P_i (of size p_i) splitting into upper and lower subsets P_i' and P_i'' (of sizes p_i' and $p_i'' = p_i - p_i'$) is

$$\frac{\binom{2^{s-1}}{p_i'}\binom{2^{s-1}}{p_i-p_i'}}{\binom{2^s}{p_i}} \quad .$$

Then

$$\text{Part}(\bar{P},\bar{R},s) \;=\; \frac{\pi\binom{2^{s-1}}{\bar{R}}\,\pi\binom{2^{s-1}}{\bar{P}-\bar{R}}}{\pi\binom{2^s}{\bar{P}}} \qquad (3.1.3)$$

and obviously $\text{Part}(\bar{P},\bar{R},s) = \text{Part}(\bar{P},\bar{P}-\bar{R},s)$, so Equation (3.1.1) becomes

$$\begin{cases} \text{for} \quad s > 0 \\ \qquad t_s(\bar{P}) \;=\; 2 \sum_{\bar{1}\le\bar{R}\le\bar{P}} \dfrac{\pi\binom{2^{s-1}}{\bar{R}}\,\pi\binom{2^{s-1}}{\bar{P}-\bar{R}}}{\pi\binom{2^s}{\bar{P}}} \cdot t_s(\bar{R}) + v\cdot\pi(1-\delta_{\bar{P},0}) \qquad (3.1.4) \\ t_0(\bar{P}) = 0 \end{cases}$$

(Notice that the lower bound $\bar{1}$ in the sum is a consequence of Equation (3.1.2).)

A further simplification can be obtained by defining

$$T_s(\bar{P}) \;=\; \pi\binom{2^s}{\bar{P}} \cdot t_s(\bar{P}) \qquad (3.1.5)$$

yielding

$$\begin{cases} \text{for } s > 0 \\ \\ T_s(\bar{P}) = 2 \sum_{\bar{1} \le \bar{R} \le \bar{P}} \pi \binom{2^{s-1}}{\bar{P}-\bar{R}} T_{s-1}(\bar{R}) + v \cdot \pi \binom{2^s}{\bar{P}} \cdot \pi(1-\delta_{\bar{P},0}) \\ \\ T_0(\bar{P}) = 0 \ . \end{cases} \tag{3.1.6}$$

Now Equation (3.1.6) can be solved by means of the generating function

$$\mathcal{T}_s \equiv \mathcal{T}_s(\bar{z}) = \sum_{\bar{P}} T_s(\bar{P}) \, \pi(\bar{z}^{\bar{P}}) \tag{3.1.7}$$

where $\bar{z}$ denotes the "m-variable" $\langle z_1, z_2, \ldots, z_m \rangle$ and thus

$$\pi(\bar{z}^{\bar{P}}) = z_1^{p_1} \cdot z_2^{p_2} \cdot \ldots \cdot z_m^{p_m} \ .$$

So

$$\mathcal{T}_s = \sum_{\bar{1} \le \bar{P}} 2 \sum_{\bar{1} \le \bar{R} \le \bar{P}} \pi \binom{2^{s-1}}{\bar{P}-\bar{R}} T_{s-1}(\bar{R}) \, \pi(\bar{z}^{\bar{P}})$$

$$+ v \cdot \sum_{\bar{1} \le \bar{P}} \pi \binom{2^{s-1}}{\bar{P}} \pi(\bar{z}^{\bar{P}}) \ . \tag{3.1.8}$$

Defining $A_s = \sum_{\bar{P}} \pi \binom{2^s}{\bar{P}} \pi(\bar{z}^{\bar{P}}) = [\pi(1+\bar{z})]^{2^s} = \bar{Y}^{2^s}$, where

$\bar{Y} = \pi(1+\bar{z})$, and noticing the convolution in the first sum of (3.1.8):

$$\mathcal{T}_s = 2 \, A_{s-1} \cdot \mathcal{T}_{s-1} + v \cdot B_s$$

where

$$B_s = \sum_{1 \le \bar{P}} \pi \binom{2^s}{\bar{P}} \pi(\bar{z}^{\bar{P}}) \ .$$

It is now a matter of iterating Equation (3.1.9),

$$\begin{aligned}
\mathcal{T}_s &= 2\,A_{s-1}(2\,A_{s-2}\,\mathcal{T}_{s-2} + v\cdot B_{s-1}) + v\,B_s \\
&= 2^2\,A_{s-1}\,A_{s-2}\,\mathcal{T}_{s-2} + 2v\,A_{s-1}\,B_{s-1} + v\,B_s \\
&\quad \cdots \\
&= 2^k\,A_{s-1}\,A_{s-2}\cdots A_{s-k}\,\mathcal{T}_{s-k} + v \sum_{0\le j<k} 2^j\,A_{s-1}\cdots A_{s-j}\,B_{s-j} \\
&= 2^{k+1}\,A_{s-1}\,A_{s-2}\cdots A_{s-k}\,A_{s-k-1}\,\mathcal{T}_{s-k-1} + 2^k\,v\,A_{s-1}\cdots A_{s-k}\,B_{s-k} \\
&\qquad + v \sum_{0\le j<k} 2^j\,A_{s-1}\cdots A_{s-j}\,B_{s-j}\ .
\end{aligned}$$

And now, recalling that $\mathcal{T}_0 = 0$,

$$\mathcal{T}_s = v\cdot \sum_{0\le j<s} 2^j\,A_{s-1}\cdots A_{s-j}\,B_{s-j}\ . \tag{3.1.10}$$

But

$$A_{s-1}\cdots A_{s-j} = Y^{2^{s-1}}\cdots Y^{2^{s-j}} = Y^{2^s-2^{s-j}} = \pi\left[(1+\bar{z})^{2^s-2^{s-j}}\right]$$

and

$$B_{s-j} = \pi \sum_{1\le \bar{P}} \binom{2^{s-j}}{\bar{P}} \bar{z}^{\bar{P}} = \pi\left[(1+\bar{z})^{2^{s-j}} - 1\right]$$

giving

$$\mathcal{T}_s = v \cdot \sum_{0 \le j < s} 2^j \cdot \pi\left[(1+\bar{z})^{2^s} - (1+\bar{z})^{2^{s-j}} \right]$$

$$= v \cdot \sum_{0 \le j < s} 2^j \; \pi\left[\sum_{\bar{P}} \binom{2^s}{\bar{P}} \bar{z}^{\bar{P}} - \sum_{\bar{P}} \binom{2^{s-j}}{\bar{P}} \bar{z}^{\bar{P}} \right]$$

$$= v \cdot \sum_{\bar{P}} \sum_{0 \le j < s} 2^j \, \pi\left[\binom{2^s}{\bar{P}} - \binom{2^{s-j}}{\bar{P}} \right] \pi(\bar{z}^{\bar{P}}) \quad . \qquad (3.1.11)$$

That yields

$$\begin{cases} T_s(\bar{P}) = v \sum_{0 \le j < s} 2^j \, \pi\left[\binom{2^s}{\bar{P}} - \binom{2^{s-j}}{\bar{P}} \right] \\ \text{and} \\ t_s(\bar{P}) = \dfrac{v}{\pi\binom{2^s}{\bar{P}}} \sum_{0 \le j < s} 2^j \, \pi\left[\binom{2^s}{\bar{P}} - \binom{2^{s-j}}{\bar{P}} \right] \quad . \end{cases} \qquad (3.1.12)$$

Equation (3.1.12) is a solution to the original recurrence, but not a very useful one. Not only does it not yield a more reasonable closed form or an asymptotic expansion, but it is very difficult to compute, due to the explosive combinatorial terms it contains!

Hence, something has to be done.

3.2 The Infinite Universe Model.

We will now analyze the behavior of the intersection for the limiting case universe $U = [0,1)$.

Let $t(\bar{P})$ be the average running time of the intersection algorithm for random subsets $P_1, P_2, \ldots, P_m$, of sizes $\bar{P} = \langle p_1, p_2, \ldots, p_m \rangle$ respectively.

No simple argument could be found to prove formally the rather intuitive fact

$$t_s(\bar{P}) < t(\bar{P}) \quad .$$

(The fact that the average number of nodes of a trie representing a random set grows with s (see Section A.2 for a detailed argument) strongly supports this intuition).

In any case it is easy to see that the probability $\mathrm{Part}(\bar{P},\bar{R},s)$ (as defined in Equation (3.1.1)) of a partition tends to the limit case

$$\lim_{s \to \infty} \mathrm{Part}(\bar{P},\bar{R},s) = \mathrm{Part}(\bar{P},\bar{R})$$

where $\mathrm{Part}(\bar{P},\bar{R})$ is the probability of a partition in the infinite universe U (refer to Equations (A.2.9) and (A.2.10)).

Consequently

$$\lim_{s \to \infty} t_s(\bar{P}) = t(\bar{P})$$

and we shall adopt $t(\bar{P})$ as an estimate of $t_s(\bar{P})$ without further discussion.

In order to compute $t(\bar{P})$ we observe that the intersection will proceed by considering both halves of U : $U' = [0,1/2)$ and $U'' = [1/2,1)$. But both U' and U'' are similar to U in the sense that the partition probabilities have the same distribution in the three cases.

Hence, by looking at all possible partitions, we arrive at the following recurrence:

$$t(\bar{P}) = \sum_{\bar{0} \leq \bar{R} \leq \bar{P}} \mathrm{Part}(\bar{P},\bar{R})[t(\bar{R}) + t(\bar{P}-\bar{R})] + v \qquad (3.2.1)$$

where v is the time required to visit the m roots of the tries.

The boundary condition is simply

$$\text{for } \bar{P}\text{: } \exists i,\ p_i = 0 \quad t(\bar{P}) = 0 \ . \tag{3.2.2}$$

And the probability of a random set $P_i \subset U$, $|P_i| = p_i$ to be of the form

$$|P_i \cap U'| = r_i \quad \text{is} \quad \frac{\binom{p_i}{r_i}}{2^{p_i}}$$

and thus

$$\text{Part}(\bar{P},\bar{R}) = \pi\binom{\bar{P}}{\bar{R}} \cdot 2^{-\Sigma\bar{P}} = \pi\binom{\bar{P}}{\bar{P}-\bar{R}} \cdot 2^{-\Sigma\bar{P}} \ . \tag{3.2.3}$$

So Equation (3.2.1) becomes

$$t(\bar{P}) = 2 \sum_{\bar{1} \le \bar{R} \le \bar{P}} 2^{-\Sigma\bar{P}}\, \pi\binom{\bar{P}}{\bar{R}} \cdot t(\bar{R}) + v \cdot \pi(1-\delta_{\bar{P}0}) \ . \tag{3.2.4}$$

An exponential generating function will take care of Equation (3.2.4):

$$\begin{aligned} \mathcal{T} = \mathcal{T}(\bar{z}) &= \sum_{\bar{P}} t(\bar{P})\, \pi\left(\frac{\bar{z}^{\bar{P}}}{\bar{P}!}\right) \\ &= 2 \sum_{\bar{P}} \left[\sum_{\bar{1} \le \bar{R} \le \bar{P}} \pi\binom{\bar{P}}{\bar{R}} \cdot t(\bar{R}) \right] \cdot \pi\left(\frac{\bar{z}^{\bar{P}}}{2^{\bar{P}} \cdot \bar{P}!}\right) \\ &\quad + v \cdot \sum_{\bar{1} \le \bar{P}} \pi\left(\frac{\bar{z}^{\bar{P}}}{\bar{P}!}\right) \ . \end{aligned} \tag{3.2.5}$$

The first term of the previous equation results from the convolution of

$$e^{\frac{\bar{1}}{2}\Sigma\bar{z}} = \sum_{\bar{P}} \pi\left(\frac{\bar{z}^{\bar{P}}}{2^{\bar{P}} \cdot \bar{P}!}\right) \quad \text{and} \quad \mathcal{T}(\bar{1}/2\ \bar{z}) \ .$$

So

$$\mathcal{T}(\bar{z}) = 2 \cdot e^{\frac{1}{2}\Sigma\bar{z}} \cdot \mathcal{T}\left(\frac{1}{2}\,\bar{z}\right) + v \cdot \pi(e^{\bar{z}} - 1) \quad . \tag{3.2.6}$$

Iterating (3.2.6)

$$\mathcal{T}(\bar{z}) = 2^k\, e^{(1-2^{-k})\Sigma\bar{z}} \cdot \mathcal{T}(2^{-k}\bar{z}) + v \cdot \sum_{0 \le j < k} e^{(1-2^{-j})\Sigma\bar{z}}\, 2^j\, \pi(e^{2^{-j}\bar{z}} - 1) \quad . \tag{3.2.7}$$

The first term of (3.2.7) vanishes with $k \to \infty$, since

$$\lim_{k\to\infty} 2^k\, e^{(1-2^{-k})\Sigma\bar{z}}\, \mathcal{T}(2^{-k} \cdot \bar{z})$$

$$= e^{\Sigma\bar{z}} \cdot \lim_{k\to\infty} \left[2^k \cdot \sum_{\bar{P}} \frac{t(\bar{P})}{2^{k\Sigma\bar{P}}} \cdot \pi\left(\frac{\bar{z}^{\bar{P}}}{\bar{P}!}\right) \right] \tag{3.2.8}$$

which goes to zero at least as fast as $e^{\Sigma\bar{z}} \cdot \mathcal{T}(\bar{z}) \cdot (1/2^k)^{m-1}$ given that $\Sigma\bar{P} \ge m$ for any $t(\bar{P}) \neq 0$. Thus we have obtained

$$\mathcal{T}(\bar{z}) = v \cdot \sum_{0 \le j} e^{(1-2^{-j})\Sigma\bar{z}} \cdot 2^j \cdot \pi(e^{2^{-j}\bar{z}} - 1)$$

$$= v \cdot e^{\Sigma\bar{z}} \cdot \sum_{0 \le j} 2^j \cdot \pi(1 - e^{-2^{-j}\bar{z}}) \quad . \tag{3.2.9}$$

The sum in (3.2.9) is

$$\sum_{0\le j} 2^j \cdot \pi(1 - e^{-2^{-j}\bar{z}})$$

$$= \sum_{0\le j} 2^j(-1)^m \cdot \pi(e^{-2^{-j}\bar{z}} - 1)$$

$$= \sum_{0\le j} 2^j(-1)^m \cdot \sum_{\bar{1}\le\bar{R}} \frac{1}{(-2^j)^{\Sigma\bar{R}}} \cdot \pi\left(\frac{\bar{z}^{\bar{R}}}{\bar{R}!}\right)$$

$$= \sum_{\bar{1}\le\bar{R}} \pi\left(\frac{\bar{z}^{\bar{R}}}{\bar{R}!}\right) (-1)^{\Sigma\bar{R}+m} \cdot \sum_{0\le j} \left(\frac{1}{2^{\Sigma\bar{R}-1}}\right)^j$$

$$= \sum_{1\le\bar{R}} \pi\left(\frac{\bar{z}^{\bar{R}}}{\bar{R}!}\right) (-1)^{\Sigma\bar{R}+m} \cdot \frac{1}{1 - 2^{-\Sigma\bar{R}+1}} \qquad (3.2.10)$$

Performing the convolution of (3.2.10) with $e^{\Sigma\bar{z}}$ we finally obtain

$$t(\bar{P}) = v \cdot \sum_{\bar{1}\le\bar{R}} \pi\binom{\bar{P}}{\bar{R}} \cdot \frac{(-1)^{\Sigma\bar{R}+m}}{1 - 2^{-\Sigma\bar{R}+1}} \quad . \qquad (3.2.11)$$

Yet another horrible expression, but this time good enough to yield some asymptotic results.

3.3 Asymptotic Behavior.

The function

$$U_n = \sum_{k \geq 2} \binom{n}{k} \frac{(-1)^k}{2^{k-1} - 1} \tag{3.3.1}$$

that appears very often in the study of binary tries will be the basis for the asymptotic expansion of Equation (3.2.11), since it is possible to express $t(\bar{P})$ in terms of U_n.

First we need to get rid of the products of combinatorial numbers in Equation (3.2.11) by noticing that for any $\bar{P}$ and x

$$\sum_j x^j \sum_{\substack{\Sigma\bar{J} = j \\ 1 \leq \bar{J}}} \pi \binom{\bar{P}}{\bar{J}} = [(1+x)^{p_1} - 1] \ldots [(1+x)^{p_m} - 1]$$

$$= \sum_{\bar{0} \leq \bar{\epsilon} \leq \bar{1}} (1+x)^{\Sigma\bar{\epsilon}\bar{P}} (-1)^{m+\Sigma\bar{\epsilon}}$$

$$= \sum_{\bar{0} \leq \bar{\epsilon} \leq \bar{1}} (-1)^{m+\Sigma\bar{\epsilon}} \sum_j \binom{\Sigma\bar{\epsilon}\bar{P}}{j} x^j$$

$$= \sum_j x^j \sum_{\bar{0} \leq \bar{\epsilon} \leq \bar{1}} \binom{\Sigma\bar{\epsilon}\bar{P}}{j} (-1)^{m+\Sigma\bar{\epsilon}} . \tag{3.3.2}$$

[Notice that the index $\bar{\epsilon}$ in $\bar{0} \leq \bar{\epsilon} \leq \bar{1}$ ranges through all the m-tuples of 0's and 1's.]

Looking at the first and last expressions in (3.3.2) it is easy to see that

$$\sum_{\substack{\Sigma\bar{J}=j \\ 1\le\bar{J}}} \pi\binom{\bar{P}}{\bar{J}} = \sum_{\bar{0}\le\bar{\epsilon}\le\bar{1}} \binom{\Sigma\bar{\epsilon}\bar{P}}{j}(-1)^{m+\Sigma\bar{\epsilon}} \quad . \tag{3.3.3}$$

Now we can massage (3.2.11) into better shape:

$$t(\bar{P}) = v\cdot \sum_{\bar{1}\le\bar{R}} \pi\binom{\bar{P}}{\bar{R}} \frac{(-1)^{\Sigma\bar{R}+m}}{1-2^{-\Sigma\bar{R}+1}} = v\cdot\sum_{m\le r} \frac{(-1)^{r+m}}{1-2^{-r+1}} \sum_{\substack{\bar{1}\le\bar{R} \\ \Sigma\bar{R}=r}} \pi\binom{\bar{P}}{\bar{R}}$$

$$= v\cdot \sum_{m\le r} (-1)^{r+m} \sum_{\substack{\bar{1}\le\bar{R} \\ \Sigma\bar{R}=r}} \pi\binom{\bar{P}}{\bar{R}} + v\cdot\sum_{m\le r} \frac{(-1)^{r+m}}{2^{r-1}-1} \sum_{\substack{\bar{1}\le\bar{R} \\ \Sigma\bar{R}=r}} \pi\binom{\bar{P}}{\bar{R}} \quad . \tag{3.3.4}$$

The first sum is Equation (3.3.2) with $x = -1$ and an extra factor of $v\cdot(-1)^m$, hence its value is v.

The second sum's index, r, ranges from m, but since the inner sum is zero for $2 \le r < m$,

$$\frac{1}{v}t(\bar{P}) = 1 + \sum_{2\le r} \frac{(-1)^{r+m}}{2^{r-1}-1} \sum_{\substack{\bar{1}\le\bar{R} \\ \Sigma\bar{R}=r}} \pi\binom{\bar{P}}{\bar{R}}$$

$$= 1 + \sum_{2\le r} \frac{(-1)^{r+m}}{2^{r-1}-1} \sum_{\bar{0}\le\bar{\epsilon}\le\bar{1}} \binom{\Sigma\bar{\epsilon}\bar{P}}{r}(-1)^{m+\Sigma\bar{\epsilon}}$$

$$= 1 + \sum_{0\le\epsilon\le 1} (-1)^{\Sigma\bar{\epsilon}} \sum_{2\le r} \frac{(-1)^{r}}{2^{r-1}-1} \binom{\Sigma\bar{\epsilon}\bar{P}}{r} \quad .$$

Thus

$$t(\bar{P}) = v \cdot \sum_{\bar{0} \le \bar{\epsilon} \le \bar{1}} (-1)^{\Sigma\bar{\epsilon}} U_{\Sigma\bar{\epsilon}\bar{P}} + v \quad . \tag{3.3.5}$$

Now Equation (3.3.5) can be expanded using

$$U_n = n[\lg n + f(n)] + cn + O(1) \tag{3.3.6}$$

where $f(n)$ is a strange periodic function of "average value" zero (see Section A.3 for further details) such that

$$|f(n)| < f_0 < 2 \times 10^{-7} \quad .$$

So, discarding the lower bound $\bar{\epsilon} = \bar{0}$ in (3.3.5)

$$t(P) = v \cdot \sum_{\bar{0} \neq \bar{\epsilon} \le \bar{1}} (-1)^{\Sigma\bar{\epsilon}} \Sigma\bar{\epsilon}\bar{P}[\lg \Sigma\bar{\epsilon}\bar{P} + f(\Sigma\bar{\epsilon}\bar{P})] + O(1) \tag{3.3.7}$$

since $c \sum_{\bar{0} \le \bar{\epsilon} \le \bar{1}} (-1)^{\Sigma\bar{\epsilon}} \Sigma\bar{\epsilon}\bar{P}$ cancels itself out.

Equation (3.3.7) provides us with the means to visualize the behavior of trie intersection. To account for the $f(\Sigma\bar{\epsilon}\bar{P})$ terms, we first notice that

$$\left| \sum_{\bar{0} \le \bar{\epsilon} \le \bar{1}} (-1)^{\Sigma\bar{\epsilon}} \Sigma\bar{\epsilon}\bar{P} f(\Sigma\bar{\epsilon}\bar{P}) \right| < f_0 \cdot \sum_{\bar{0} \le \bar{\epsilon} \le \bar{1}} \Sigma\bar{\epsilon}\bar{P} = f_0 \cdot 2^{m-1} \Sigma\bar{P} \tag{3.3.8}$$

which shows that, even without considering cancellations, we can safely ignore the contribution of the $f(\Sigma\bar{\epsilon}\bar{P})$ terms when intersecting sets of size $O(1/f_0)$, and thus

$$t(\bar{P}) = v \cdot \sum_{\bar{0} \neq \bar{\epsilon} \le \bar{1}} (-1)^{\Sigma\bar{\epsilon}} \Sigma\bar{\epsilon}\bar{P} \lg \Sigma\bar{\epsilon}\bar{P} + O(\text{small}) \quad , \tag{3.3.9}$$

where $O(\text{small})$ stands for the periodic terms that we will henceforth discard.

3.4 Overall Bounds on Trie Intersection.

Equation (3.3.9) has finally provided us with a more manageable bound on the "wasted effort" for the trie intersection algorithm. But we still have to estimate the time $V(\bar{P}, q)$ spent verifying that an arbitrary intersection $Q = P_1 \cap P_2 \cap \ldots \cap P_m$ is a proper subset of each set.

Consider a set P and a proper subset $Q \subseteq P$; Figure 3.4.1 shows the binary trie representation for P , the shaded nodes would also appear in the trie for Q .

In order to verify that $Q \subseteq P$ we need to

- traverse all the shaded nodes (the trie for Q)
- every time that the boundary is reached we need to follow a path to the element in P that belongs in Q .

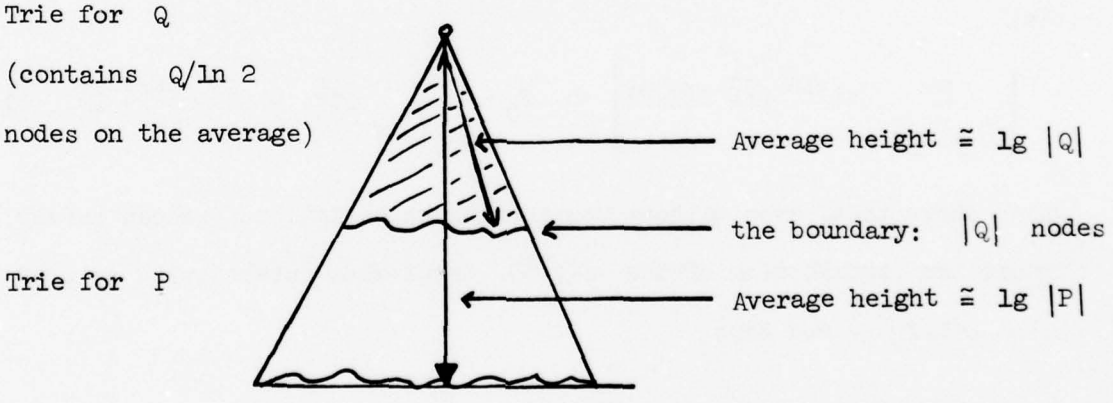


Figure 3.4.1. A trie for a set P and a proper subset $Q \subseteq P$.

Thus we can estimate that the average time $\nu(p,q)$ for random set P, $|P| = p$, and subset $Q \leq P$, $|Q| = q$ is

$$\nu(p,q) = \frac{q}{\ln 2} + q(\lg p - \lg q)$$

$$= q\left(\frac{1}{\ln 2} + \lg \frac{p}{q}\right) \qquad (3.4.1)$$

yielding

$$V(\bar{P},q) = \sum_i \nu(p_i,q) = m \cdot \frac{q}{\ln 2} + q \sum_i \lg(p_i/q) \quad . \qquad (3.4.2)$$

Combining Equations (3.3.9) and (3.4.2) results in

$$I(\bar{P},q) \leq v \cdot \sum_{\bar{0} \neq \bar{\epsilon} \leq \bar{1}} (-1)^{\Sigma\bar{\epsilon}} \Sigma\bar{\epsilon}\bar{P} \lg \Sigma\bar{\epsilon}\bar{P}$$

$$+ q\left[\frac{m}{\ln 2} + \sum_i \lg(p_i/q)\right] \qquad (3.4.3)$$

our final bound for the trie intersection algorithm.

Lastly, let us discuss a couple of particular cases.

Intersection of two sets: Equation (3.4.3) becomes

$$I(p_1,p_2,q) \leq v\, p_1 \lg\left(1 + \frac{p_2}{p_1}\right) + v\, p_2 \lg\left(1 + \frac{p_1}{p_2}\right)$$

$$+ q\left(\frac{2}{\ln 2} + \lg \frac{p_1 \cdot p_2}{q^2}\right) \quad . \qquad (3.4.4)$$

It is interesting to analyze the "wasted effort" $I(p_1,p_2,0)$ separately:

for $p_1 \ll p_2$: $\quad I(p_1,p_2,0) = O(p_1 \lg p_2)$

for $p_1 \approx p_2$: $\quad I(p_1,p_2,0) = O(p_1 + p_2)$

for $p_1 \gg p_2$: $\quad I(p_1,p_2,0) = O(p_2 \lg p_1)$.

In other words, when the sets are different in size, trie intersection behaves like a series of binary searches and as they become comparable in size, the intersection time becomes the usual ordered list intersection time.

Intersection of sets of equal size: Assume $\bar{P} = \bar{p}$, and that the time v to visit a given node is simply m, then

$$I(\bar{p},q) \leq p \cdot m \cdot L_m + q \cdot m \cdot \left[\frac{1}{\ln 2} + \lg \frac{p}{q} \right] \qquad (3.4.5)$$

where

$$L_m = \sum_{j \geq 2} \binom{m}{j} (-1)^j \, j \lg j \quad .$$

The factor L_m does not seem to decrease fast enough (see Table 3.4.2 below) to make the parallel computation of the intersection competitive with a sequence of intersections (intersect $P_1 \cap P_2$, then $(P_1 \cap P_2)$ with P_3, and so on) that would very quickly "weed out" the elements that do not belong in Q. But three-way intersection may be better than two.

m	L_m	$m \cdot L_m$
2	2.0	4.00
3	1.25	3.74
4	0.98	3.92
5	0.84	4.21
10	0.58	5.85
20	0.45	8.98

Table 3.4.2. Tabulation of L_m.

4. Hashed Representation and Intersection of Sets.

Searching for an element in a given set can be a very fast operation; this suggests the following intersection procedure:

> Let P and R be the sets to be intersected. Choose the smallest one (say P), and search for each one of its elements in R.

The average time to intersect the above two sets is

$$t(P,R) = C(R) \cdot |P| \qquad (4.0.1)$$

where $C(R)$ is the average time to search for an element in R. And every practitioner of the arcane mysteries of computing knows that hashing methods require (on the average) only a constant time per search, so the intersection of two sets P and R of sizes p and r will require

$$t(p,r) = c \cdot \min(p,r) \quad . \qquad (4.0.2)$$

So hashed intersection, as we shall call the intersection method loosely described above, seems difficult to beat: its running time is proportional to the size of the smallest set and the constant c can be made duly small by choosing a good enough hashed representation.

But not everything is rosy: accessing a hash table is usually done in a random fashion, thus requiring a large working set for the algorithm; further, the worst case running time can be very bad (as bad as $O(p \cdot q)$).

The first part of this chapter deals with different hashed representations. The second part refers to the hashed intersection algorithm, especially to the problems that we have just stated, presenting solutions to them and discussing some of the trade-offs involved.

4.1 Storing the Sets.

This section discusses different ways of representing unchanging sets utilizing hashing techniques.

In order to represent a set $S \subseteq U$, we need

- a hash array $T = \langle t_0, t_1, \ldots, t_{M-1} \rangle$
- a hash function $h : U \rightarrow [0,M)$;

and
- an unhash function $h^{-1} : [0,M) \times T \rightarrow U$.

Now the elements of S are hashed into T by means of the hash function h and a suitable collision resolution strategy. Enough information about each element is stored in T so that further retrieval is possible by means of the unhash function h^{-1} .

We will further assume that all the hash functions chosen are probabilistically "fair", in the sense that for all $x,y \in U$ and for all $\alpha \in [0,M)$: $\text{Prob}[h(x) = \alpha] = \text{Prob}[h(y) = \alpha] = 1/M$.

The problem of optimal storage of unchanging sets by means of hashing techniques has very seldom been addressed (see [Greniewski and Turski 63] and [Sprugnoli 77]). One possible reason for this absence of interest is that normal hashing techniques are reasonably good for static sets.

We will nevertheless investigate some basic techniques and discuss some of the extra advantages resulting from the availability of preprocessing.

Throughout this section and as a means of comparison we will use the same Example 2.0.1 used in Chapter 2. We will also refer to a universe $U_s = [0,2^s)$ and a subset $S = \{s_1, s_2, \ldots, s_n\} \subseteq U_s$. Also, given a hash function h , $\text{hash}(h,S) = \langle h(s_1), h(s_2), \ldots, h(s_n) \rangle$ will be the hash sequence of S .

Again we will be interested in measuring the number of bits $B_i(s,n)$ that representation i requires for random sets of size n. Also, the average retrieval times $R_i(s,n)$ (for a successful search) and $R_i'(s,n)$ (unsuccessful search) will be presented. Retrieval time will be measured in terms of the number of hash function evaluations and/or comparisons needed.

The Basic Representation.

For fast retrieval and simplicity, nothing beats hashing with collision resolution by separate chaining (see [Knuth 75, Section 6.4]). The i-th entry t_i of the hash table T is of the form

$$t_i = \langle tag_i , v_i , link_i \rangle$$

where

tag_i is occupied if $\exists x \in S: i = h(x)$,
is free otherwise;

v_i is an element of the set S (or a special empty value);

$link_i$ points to the next element in the chain that hashes to $h(v_i)$ (or it is a special null value).

(Figure 4.1.1 shows the sample set represented in this fashion.)

The size of the table M and the size of the set $|S| = n$ determine the load factor

$$\alpha = n/M \quad . \tag{4.1.1}$$

So we have

$$B_1(s,n) = M(s + \lg M + 1)$$

$$= n \cdot \frac{1}{\alpha}(s + \lg n - \lg \alpha + 1) \quad . \tag{4.1.2}$$

And the retrieval times are

$$R_1(s,n) = 1 + \frac{\alpha}{2} - \frac{1}{2M}$$

$$R_1'(s,n) = e^{-\alpha} + \alpha + O(M^{-1}) \quad . \tag{4.1.3}$$

The above retrieval times could be improved by sorting the collision chains but the improvement is minimal. (Notice that even with a full table, $\alpha = 1$, retrieval requires fewer than 1.5 probes on the average.)

A More Economical and Flexible Representation.

The fact that an element hashes to a given address provides information about that element that can be used by the unhash function, thus requiring less information to be stored in the v -fields of the table. Roughly speaking, the hashing address provides us with $\lg M$ bits of information and only about $(s - \lg M)$ bits that need to be stored. (This idea is due to Butler Lampson, see [Knuth 75, p. 518].)

For instance, if the hash function is $h(x) = x \bmod M$, then we only need to store $\lfloor x/M \rfloor$, since $x = \lfloor x/M \rfloor + x \bmod M$, and $\lfloor x/M \rfloor$ requires only $(s - \lg M)$ bits.

There is also a useful trade-off between space and retrieval time: by separating the hash table and the value fields and storing the values sorted by hash address we can vary the size of the hash table at will, changing the retrieval times correspondingly.

Two tables are needed: the hash table proper, T, with entries

$$t_i = \langle link_i \rangle \qquad 0 \le i \le M$$

$link_i$ points to the value table (if the entry is free, $link_i$ points to the first value of the chain corresponding to the next occupied entry, thus $link_i = link_{i+1}$)

and a value table, V, $|V| = n$,

$$v_k = \langle r_k \rangle$$

r_k is such that if $h(x) = i$ and $link_i \le k < link_{i+1}$, then $h^{-1}(i, r_k) = x$

(see Figure 4.1.2 for an example).

The above representation makes unhashing of a single element a rather difficult task (since given an entry k in V, the corresponding hash address i must be figured out by searching T for $link_i \le k < link_{i+1}$), but the intersection algorithm requires unhashing of whole sets only, and that is easily accomplished in $O(n)$ time.

The ordering of the v fields, a simple preprocessing task, is important for two reasons: first, it eliminates the need for pointers in the V table and, second, it cuts the unsuccessful search time by about half, since the search in the collision table is now a search on an ordered list.

The space requirements are

$$B_2(s,n) = M(\lg n) + n[\, s - \lg M] = n\left[s - (\lg n)\left(1 - \frac{1}{\alpha}\right) + \lg \alpha \right] . \qquad (4.1.4)$$

i	tag_i	v_i	$link_i$
0	occ.	189	null
1	occ.	130	null
2	free	232	null
3	occ.	3	null
4	occ.	25	
5	occ.	200	null
6	occ.	51	
7	free	249	null
8	occ.	50	null

$M = n = 9 \qquad \alpha = 1$

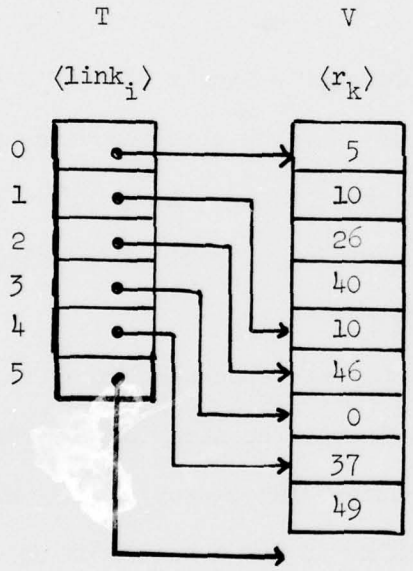
$h(x) = (97x) \bmod 9$

$h^{-1}(i) = v_i$

$B_1(8,9) = (1+8+4) \times 9$

$= 117$ bits

Figure 4.1.1: Hashing with separate chains.



$M = 5$

$\alpha = 9/5$

$h(x) = x \bmod 5$

$h^{-1}(i,k) = 5r_k + i$

$B_2(8,9) = (5+1) \times 3$ (for T)

$+ 9 \times 6$ (for V)

$= 72$ bits

Figure 4.1.2: Separate hash and value tables; abbreviated value fields.

The retrieval times count the average number of accesses to both T and V, and are

$$R_2(s,n) = 2 + \frac{\alpha}{2} + O(M^{-1})$$
$$R'_2(s,n) = 2 + \frac{\alpha}{2} - \frac{1-e^{-\alpha}}{\alpha} + O(M^{-1}) \qquad (4.1.5)$$

(see [Knuth 73], Exercise 6.4-35).

Coping With Worst Cases: Persistent Hashing.

Everybody knows that one of the best ways to punish a computer scientist is to make her/him meditate about the incredibly horrendous worst case of hashing, without the protection of probabilities. So it might seem that little can be done about worst cases, but a recent result in probability theory together with the availability of preprocessing helps in the case of static sets.

Hashing fits into the traditional probabilistic model of throwing n balls into M urns (the hash table), each ball falling into any urn with uniform probability $1/M$. The hash function acts as a randomizing agent that distributes the n elements of the set more or less uniformly. The problem is that the maximum number of balls falling into an urn can be as high as n. Here is where a result that follows from research done by Persi Diaconis and David Freeman [Diaconis and Freeman 78] is useful.

Theorem D. In the occupancy model defined above, let $\alpha = n/M$ and $\alpha \ll \lg n$. Let $M_0(n)$ be the maximum number of balls falling into any single urn, then

$$\lim_{n \to \infty} \text{Prob}[0 \leq M_0(n) - m^*(n,\alpha) \leq 2] = 1$$

where $m^*(n,\alpha) = O\left(\frac{\ln n}{\ln \ln n}\right)$. □

In short, the worst case of hashing is almost always going to be less than the logarithm of n .

But what happens if not? For example, a hash function like $h(x) = x \bmod M$ yields a worst case of n for sets of the form $S = \{a + iM \mid 0 \leq i < n\}$. The answer is: change the hash function. If we have a good set of hash functions we can simply keep trying until we hit one that is good enough.

And thus, persistent hashing: given a family H of hash functions and a set S , randomly choose a hash function $h \in H$. Now hash the set S with h and check for the worst case: if this turns out to be worse than $\ln n$ then pick another $h' \in H$ at random and repeat the procedure until the worst case condition is satisfied.

Given the above procedure, we could assign the "truly randomizing" blue ribbon to the family H if, for any set S , independently random hash sequences $\text{hash}(h_1,S), \text{hash}(h_2,S), \ldots$ are produced where the hash functions $h_1, h_2, \ldots$ are randomly chosen within H . Notice that in this case the expected number of trials (of different hash functions) would be just 1 .

Further research is required to characterize the behavior of practical hash families with respect to persistent hashing. (J. L. Carter and Mark N. Wegman [Carter and Wegman 77], developed a related idea: universal classes of hash functions, for the usual case of changing sets.)

And finally, recursive hashing.

The previous paragraphs presented a technique (persistent hashing) to partition a set S into subsets of at most $\ln |S|$ elements. The recursive application of persistent hashing to each one of the partitions of S until reaching an acceptable number of collisions is what we will call recursive hashing. (A particular instance of this technique can be found in [Sprugnoli 77].)

For instance, Figure 4.1.3 shows the representation of Figure 4.1.2, with the first collision list also hashed.

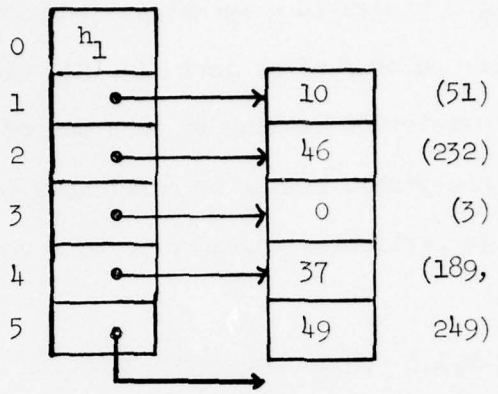
Notice that the bounds for space and average retrieval time are at least as good as the second representation considered, but the worst case can be considerably improved.

An interesting open problem is to investigate the trade-offs involved in choosing load factors for the hash tables at different levels.

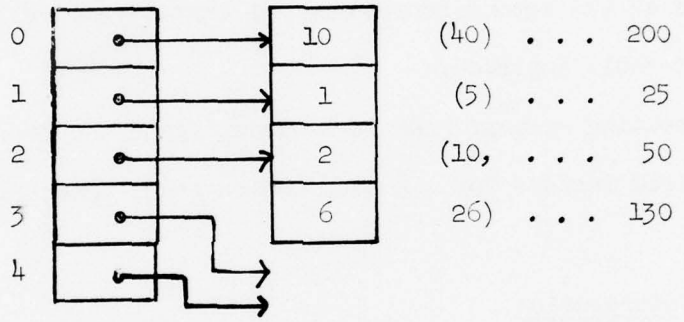
4.2 Hashed Intersection.

Time Bounds for Multiple Set Intersection.

In order to intersect m sets, we simply intersect the two smallest ones, then the resulting intersection with the third smallest, and so on. So assume the sets $P_1, P_2, \ldots, P_m$ of sizes $\bar{P} = \langle p_1, p_2, \ldots, p_m \rangle$ and that the sets are ordered by size so that $p_1 \leq p_2 \leq \cdots \leq p_m$. Let also $Q = P_1 \cap P_2 \cap \ldots \cap P_m$ be the intersection ($|Q| = q$), and $W_i = P_i - Q$ be all the elements of P_i that do not belong in the intersection.



$h(x) = x \bmod 5$ for this table: $h^{-1}(i,k) = 5r_k+i$



$h_1(y) = y \bmod 4$ for this table:

$h_1^{-1}(i,k) = (4r_k+i) \times 5 + 0 = 20r_k + 5i$

Figure 4.1.3. An example of recursive hashing.

According to our discussion in the first chapter, the average intersection time can be analyzed as

$$I(\bar{P},q) = t(\bar{P},q) + V(\bar{P},q) \qquad (4.2.1)$$

where $t(\bar{P},q)$: average wasted time (time to verify that $W_1 \cap W_2 \cap \ldots \cap W_m = \emptyset$)

and $V(\bar{P},q)$: time needed to verify that each element of Q belongs to each one of $P_1, P_2, \ldots, P_m$.

In order to evaluate $t(\bar{P},q)$ we use the hypothesis that the subsets W_i are random subsets of $U - Q$, that is:

$$\text{for all } x \in U\text{-}Q : \quad \text{Prob}[x \in W_i] = \frac{|W_i|}{|U - Q|} \quad , \qquad (4.2.2)$$

which means that the expected sizes of the intersections W_1 and W_2 will be

$$E(|W_1 \cap W_2|) = \frac{|W_1|}{|U - Q|} \times \frac{|W_2|}{|U - Q|} \times |U - Q|$$

or in general

$$E(|W_1 \cap W_2 \cap \ldots \cap W_k|) = \left(\prod_{1 \le i \le k} |W_i| \right) \cdot |U - Q|^{-k+1} \qquad (4.2.3)$$

and since

$$\frac{|W_i|}{|U - Q|} \le \frac{|W_i| + |Q|}{|U - Q| + |Q|} = \frac{|P_i|}{|U|} = \frac{p_i}{2^s}$$

we get

$$E(|W_1 \cap W_2 \cap \ldots \cap W_k|) \le 2^s \prod_{1 \le i \le k} (p_i/2^s) = p_1 \cdot \prod_{2 \le i \le k} (p_i/2^s) \, . \qquad (4.2.4)$$

So now we are ready to evaluate

$$t(\bar{P},q) = \sum_{1 \le k < m} E(|W_1 \cap W_2 \cap \ldots \cap W_k|) \cdot R'(s,p_{k+1}) \quad ; \qquad (4.2.5)$$

that is, the k-th step of the intersection requires the time needed to retrieve each element in $W_1 \cap W_2 \cap \ldots \cap W_k$ from W_{k+1},

$$t(\bar{P},q) \le C^* \cdot p_1 \cdot \sum_{1 \le k < m} \prod_{2 \le i \le k} (p_i/2^s) \qquad (4.2.6)$$

where C^* = maximum average retrieval time (successful or unsuccessful) over all sets.

The sum in (4.2.6) is easily bounded by

$$\sum_{1 \le k < m} \prod_{2 \le i \le k} (p_i/2^s) \le \sum_{1 \le k < m} (p_m/2^s)^{k-1}$$

$$\le \frac{1}{1 - p_m/2^s} \qquad (4.2.7)$$

yielding

$$t(\bar{P},q) \le C^* \cdot p_1 \cdot \frac{1}{1 - p_m/2^s} \quad . \qquad (4.2.8)$$

The time needed to verify the intersection equals the time needed to intersect Q and each P_i, thus

$$V(\bar{P},q) \le C \cdot m \cdot q \qquad (4.2.9)$$

where C = maximum average successful retrieval time over all sets.

So we obtain

$$I(\bar{P},q) \le C^* \cdot p_1 \cdot \frac{1}{1 - p_m/2^s} + C \cdot m \cdot q \quad . \qquad (4.2.10)$$

Finally, we should notice that if the sets to be intersected belong to a given family $\wp$ of sets (the indexes of a library, for instance) we can define the global maxima

$$\begin{aligned} &C(\wp): \quad \text{maximum successful retrieval time over all sets of } \wp \\ &C^*(\wp): \quad \text{idem, both successful and unsuccessful} \\ &\rho = \max_{P \in \wp} [|P| / 2^s] \end{aligned} \tag{4.2.11}$$

so that

$$I(\bar{P}, q) \leq C^*(\wp)\, p_1 \frac{1}{1-\rho} + C(\wp)\, m\, q \tag{4.2.12}$$

that is: the intersection time is <u>linear</u> on

- the size of the smallest set to be intersected;
- the product of the number of sets and the size of their intersection.

<u>The Working Set Size</u>.

In most practical cases the sets to be intersected will be stored in a hierarchical memory system (some kind of paging scheme, a conventional file structure, etc.) and it is crucial to minimize the total number of transfers between memory hierarchies (the overall working set size).

In the case of trie intersection, the algorithm runs through the tries in preorder. By sequentially storing the trie nodes in that order, a simple preprocessing task, the working set is minimized since once a given block of information (a page or a physical block in a file) is used, it will not be required again. But hash tables are accessed in a basically random fashion so that the same block of information might be required at different times when computing the intersection.

If no preventive measures are taken, the number of transfers is going to be exactly the quantity $t(\bar{P},q)$ evaluated before, Equation (4.2.10), with the constants C and C^* taking the value of the average number of transfers required for a retrieval operation.

But suppose we want to intersect the sets P_1 and P_2 (stored using hash functions h_1 and h_2): if we managed to look for the elements of P_1 in the same order that they are (or would be) sequentially stored in P_2, then we would have minimized the number of transfers involved. This idea can be utilized in two ways:

(i) Sort P_1 using the set of values $h_2(P_1)$ as sorting keys. Now, P_2 may have been preprocessed so that an orderly (according to h_2) traversal minimizes the number of transfers (for instance: the second representation considered in the previous section does it). But a solution like this introduces a sorting step with the logarithmic factor attached to it.

(ii) Make h_1 and h_2 define compatible orderings, in the sense that $h_1(x) \leq h_1(y)$ iff $h_2(x) \leq h_2(y)$. (For instance, define a unique global scrambling function $h^*: U \to [0,1)$, and if $h_i: U \to [0,M_i)$ then $h_i(x) = \lfloor h^*(x) * M_i \rfloor$.) Such a strategy eliminates the possibility of choosing a better hash function by "persistent hashing", though we still may use a scheme like "recursive hashing" with the h_i's as first level hash functions for all the sets.

There is yet another way of dealing with the working set problem: reduce the size of the data structure used to compute the intersection. A technique to do so is analyzed in the next subsection.

The Use of Hashed Maps.

A hash function $h: U \rightarrow [0,M)$ induces a partition of the universe U into M equivalence classes. Given a set S, $h(S)$ will be the h-map associated with it. (Normally $h(S)$ will be represented as a bit string of M bits.)

The important point about hashed maps is that if a given element $x \in U$, hashes into $h(S)$, that is $h(x) \in h(S)$, then the probability of x belonging to S is larger than the usual probability $|S| / |U|$.

This last fact was originally exploited by Burton H. Bloom [Bloom 70] in a method for reducing the number of accesses to secondary storage when dealing with very large hash tables. (Current research by Robert W. Floyd [Floyd 77] also deals with hashed maps.)

The intersection of two sets P_1 and P_2 may now be computed as follows:

(i) Hash the elements of P_1 using h_2. Let P_1' be the subset of P_1 that hashes into $h_2(P_2)$.

(ii) Perform the usual intersection between P_1' and P_2. (Notice that the subset $P_1 - P_1'$ that does not hash into $h_2(P_2)$, does not intersect P_2.)

If γ_2 is the probability of falsely accepting x, that is

$$\gamma_2 = \text{Prob}[h_2(x) \subseteq h_2(P_2) \text{ and } x \notin P_2]$$

then the first step will weed out a fraction $(1 - \gamma_2)$ of the elements of P_2 that do not belong in the intersection, thus reducing the work in Step (ii).

So let us now take a closer look at hashed maps. Given a set S we will associate with it a "composite hash function" h consisting of d functions $h^{(1)},h^{(2)},\ldots,h^{(d)}$ hashing into $\mathcal{M} = [0,M)$. An element x hashes into a subset of $\mathcal{M}$, the "hash set" of x:

$$h(x) = \{h^{(i)}(x) \mid 1 \leq i \leq d\} \tag{4.2.13}$$

and the hashed map of S is

$$h(S) = \bigcup_{x \in S} h(x) \quad . \tag{4.2.14}$$

(Notice that for an element to belong in S, its hash set must be included in $h(S)$:

$$\forall y \in U: \; h(y) \not\subset h(S) \Rightarrow y \notin S \quad .)$$

We estimate the probability γ of falsely accepting an element x. Let Φ be the probability of an element of $\mathcal{M}$ not to belong in $h(S)$; then

$$\Phi \approx (1 - d/M)^n \approx e^{-dn/M} \quad . \tag{4.2.15}$$

The probability of a given element x being accepted is the probability that all of $h^{(i)}(x)$ belong in $h(S)$, thus

$$\gamma \approx (1 - \Phi)^d - |S| / |U| \quad . \tag{4.2.16}$$

Now, the probability of x belonging to S, $|S| / |U|$, may be assumed for all practical purposes to be $|S| / |U| \ll \gamma$, so

$$\gamma \approx (1 - e^{-dn/M})^d = (1 - e^{-d\alpha})^d \tag{4.2.17}$$

where $\alpha = n/M$ is the load factor of the hash map.

For a given load factor α_0, there is a value of d_0 that minimizes γ to an optimum γ_0

$$\alpha_0 \, d_0 = \ln 2$$

$$\gamma_0 = (1 - e^{-\ln 2})^{d_0} = 2^{-d_0} = (0.619)^{1/\alpha_0} \quad . \tag{4.2.18}$$

For instance, with three hash functions, $d_0 = 3$, and using about 4.3 bits per element of S, $\alpha_0 = 0.23$, we get $\gamma_0 = 1/8$. If, on the other hand, only one hash evaluation is used, the probability increases to about 0.21 (still using the same load factor, $\alpha = 0.23$).

In order to estimate the time $I_\gamma(\bar{P}, q)$ needed to intersect m sets of sizes $P = \langle p_1, p_2, \ldots, p_m \rangle$, we may use exactly the same analysis that yielded $I(\bar{P}, q)$ in the first part of this section, obtaining

$$I_\gamma(\bar{P}, q) \leq C_h^*(\rho) \, p_1 \frac{1}{1-\rho} + C^*(\rho) \, p_1 \frac{\gamma}{1-\rho} + C(\rho) \, m \, q$$

$$= (C_h^*(\rho) + \gamma \, C^*(\rho)) \, p_1 \frac{1}{1-\rho} + C(\rho) \, m \, q \quad . \tag{4.2.19}$$

where $C(\rho)$ and $C^*(\rho)$ are maximum average number of retrieval tries as in (4.2.11)

and $C_h^*(\rho)$ is the average time required to check whether $h(x)$ belongs in a hashed map.

(The term $C^*(\rho) \, p_1 \, \gamma/(1-\rho)$ is a bound on the time needed to reject the elements falsely accepted by the hashed maps.)

Two important characteristics of the use of hashed maps may be singled out:

First, by checking the hashed maps first, it is possible to obtain a good estimate of the size of the intersection (the estimate will have an error of at most $p_1 \cdot \gamma^{m-1}$). This could be an important advantage

in interactive query systems (to produce answers like: "Your request covers at most 38 papers. Do you want them listed?").

Second, the hashed maps are small compared with the full set ($1/\alpha$ bits versus roughly s bits per set element), so if the right kind of hash function is chosen for the hashed maps, a sensible reduction in the number of transfers from secondary storage may be obtained. (One possibility: a unique global hash function for all the hashed maps, as discussed in the previous subsection.)

The Worst Case, Revisited.

We have already discussed the worst case of retrieval time for individual sets. Now it is time to look at the worst case when computing the intersection of our beloved m sets $P_1, P_2, \ldots, P_m$ (as usual, sizes are $\bar{P} = \langle p_1, p_2, \ldots, p_m \rangle$).

Having a bound on the worst case for retrieval yields a worst case for the intersection

$$W(\bar{P}) = \sum_{1<k\leq m} p_1 \cdot \ln p_k \leq p_1(m-1) \ln p_n \qquad (4.2.20)$$

that is still rather terrible.

One solution consists in ordering the traversal through the different sets by adopting a global hash function, and storing the collision lists in order. Assume now we want to intersect P_1 and P_2. The algorithm proceeds as usual except that a "last visited element" pointer is kept up-to-date. Whenever a new element of P_1 has to be searched for, the last visited element is checked: if it belongs in the same collision list as the element being searched for, the search starts with the "next-to-last

visited element". This scheme (extended to simultaneous intersections of m sets) guarantees that

$$W(\bar{P}) \leq \sum \bar{P} \tag{4.2.21}$$

since no element in the sets will ever be visited twice.

And (4.2.21) is not such a bad worst case (similar to the worst case for sorted sequential representation), so we close this chapter happily.

5. Conclusions.

About what has been presented.

This work presented two different methods to store and operate on sets, one based on tries and the other on hashing techniques.

In both trie and hashing based representations, the storage requirements were very good: less than what would be needed by sequentially allocating the same set and very close to the theoretically minimum requirements.

Sets were considered static and a certain amount of preprocessing time was assumed to be available. Trie building reduced mainly to a sorting process. In the case of hashed representation, some kind of sorting was used but, also, preprocessing time could be used choosing a convenient hash function from a pre-existing family.

In general the representations here presented have a very good average retrieval time (logarithmic for the trie structures and constant for the hashed representations). The worst case is the logarithm of the size of the universe for the trie structures, and by means of preprocessing, may be reduced to logarithmic in the hashed case.

Very few of the representations allowed for easy update procedures. On the assumption of sets being static, updates were supposed to be processed as a recreation of the whole set.

And finally intersection. Trie intersection turned out to have good average running times when the sizes of the sets to be intersected are different. On the other hand, the algorithm behaves like the usual ordered list intersection for sets of similar size. The algorithm proceeds by an orderly traversal through the data structure, and this fact minimizes the overall working set (provided the tries have also been stored in an orderly fashion).

Hashed intersection proved to be the fastest gun in the West: its wasted effort is truly minimal (bounded by a constant times the size of the smallest set to be intersected, and independent of the number of sets being intersected). But some problems had to be faced: a random access pattern to the data structure that could pessimize the working set, and the possibility of a horrendous worst case. Techniques were presented to get around these latter problems, though it is not clear how much the solution to the problems could affect the excellent run time.

About what has not been presented.

Perhaps the greatest absentee in this work has been the discussion of set union algorithms, and we will briefly cover them here.

Both representations permit the implementation of efficient union algorithms. It must be remarked that by union we mean "destructive" union, where one of the sets is replaced by the union of itself and other sets. (If the result is going to be generated anew, then there is not much to be gained over the ordered list union.)

In the case of trie representation, assume we wish to compute the union of two sets P_1 and P_2, and store the result in P_1. The algorithm proceeds by traversing the nodes common to both tries. When reaching a subtrie in P_2 that is empty in P_1, the algorithm simply appends it to P_1 and proceeds. This procedure has exactly the same complexity as the intersection (since only common nodes are visited). So on the average the time needed to compute the union of two sets of sizes p_1 and p_2 will be given by

$$\text{Union}(p_1,p_2) = p_1 \lg\left(1 + \frac{p_2}{p_1}\right) + p_2 \lg\left(1 + \frac{p_1}{p_2}\right) . \qquad (5.1)$$

Notice that the union modifies the data structure, so none of the preordered representations could be used. (The reader is referred to [Brown and Tarjan 77] for a more complete discussion of the union operation and another implementation of it.)

In the case of hashed representations, a union algorithm is also possible provided some sort of overflow handling representation is adopted. In this case the average time required is simply

$$\text{Union}(p_1,p_2) = c \cdot \min(p_1,p_2) \qquad (5.2)$$

but it is not clear whether the above statement is fair since the outcome of the union is a "deteriorated" set (with a larger load factor and, perhaps, a worse worst-case retrieval time). In any case, some way of revamping such sets may be needed for multiple set intersection.

<u>And now ... for something completely different: practical considerations</u>.

This work presented a set of techniques and discussed them, but before using them in the real world many choices have to be made. Let us assume for a while that we wish to implement a data base by means of inverted files.

Choosing a representation and tuning it to the computer system characteristics is the first task. Important factors for this decision are the machine addressing and bit handling facilities (do I pack nodes or do I respect byte boundaries?) and the memory hierarchy (how big a bucket? What about the expected working set size?)

A second major set of decisions is performance related. Space-time trade-offs (what load factor in a hash representation?). Or what price for updating? (batch the updates or leave some slack in the data structures and reorganize the data base every so often?) Or decide about extras (like hashed maps). Or real-time requirements (use some kind of persistent or recursive hashing in order to improve the worst-case retrieval?)

A final choice consists in the collection of statistical data to verify and improve the assumption made: How big are real-life intersections compared to the sets they come from? What does a query look like?

Enough. Let us go back to Academia.

Further Research.

Some of the problems here presented deserve further analysis.

- Characterize randomizing families of hash functions and further investigate persistent hashing.
- Investigate recursive hashing. What space-time trade-offs are worthwhile? How can the worst case be improved (assuming preprocessing)?
- Given a data base organized by means of inverted files, what is the best way to process general queries (including intersections and unions)? What kind of "planning aids" (like hashed maps) should be stored in the data base to help decide how to process a given query?

Appendix A.

This appendix investigates the asymptotic average behavior of binary tries.

Section A.1 discusses the number of nodes $A_2(s,n)$ of a trie representing a random set of n elements from a universe $[0,2^s)$. Section A.2 discusses $A_2(s,n)$ further, in particular the fact that

$$A_2(s+1,n) > A_2(s,n) \quad ,$$

and its limiting case when s grows indefinitely.

The rest of the appendix uses the universe $U = [0,1)$, and the average set within it. Section A.3 presents the basic counting equation. Section A.4 counts nodes with two nonterminal siblings (two-way branching). Section A.5 computes a bound for the space needed to store pointers $(P_5(s,n))$ in representation 5 (of Chapter 2). And finally, Section A.6 discusses the growth of $A_2(s,n)$ for increasing values of s.

A.1 Counting Nodes: $A_2(s,n)$.

Section 2.1 defines the quantity $A_2(s,n)$ as the average number of nodes of a trie representing a random set S defined by

$$|S| = n \quad , \quad S \subseteq U = [0,2^s)$$
$$\text{and} \quad \text{Prob}[S = \{\alpha_1,\alpha_2,\ldots,\alpha_n\}] = 1/\binom{2^s}{n} \qquad \text{(A.1.1)}$$
$$\text{for any} \quad \{\alpha_1,\alpha_2,\ldots,\alpha_n\} \subset U \quad .$$

The equation that defines $A_2(s,n)$ is

$$A_2(s,n) = (1-\delta_{n0}-\delta_{n1})(1-\delta_{s0}) + \sum_p \frac{\binom{2^{s-1}}{p}\binom{2^{s-1}}{n-p}}{\binom{2^s}{n}} [A_2(s-1,p) + A_2(s-1,n-p)] \qquad (A.1.2)$$

(since $\binom{2^{s-1}}{n}\binom{2^{s-1}}{n-p}\Big/\binom{2^s}{n}$ is the probability of a partition $(p,n-p)$).

Defining

$$B(s,n) = \binom{2^s}{n} A_2(s,n) \quad , \qquad (A.1.3)$$

and given the symmetry of (A.1.2),

$$B(s,n) = \binom{2^s}{n}(1-\delta_{n0}-\delta_{n1})(1-\delta_{s0}) + 2\sum_{p\geq 0}\binom{2^{s-1}}{n-p}B(s-1,p) \; . \qquad (A.1.4)$$

The generating function

$$\begin{cases} B_s(z) \equiv B_s = \sum_n B(s,n) z^n \\ \qquad = \sum_{n\geq 2}\binom{2^s}{n} z^n + 2\sum_n \left[\sum_{p\geq 2}\binom{2^{s-1}}{n-p} B(s-1,p)\right] z^n \\ B_0(z) \equiv B_0 = 0 \end{cases} \qquad (A.1.5)$$

can be expressed in terms of

$$\begin{aligned} G_s(z) \equiv G_s &= \sum_{n\geq 2}\binom{2^s}{n} z^n = (1+z)^{2^s} - 1 - 2^s z \\ F_s(z) \equiv F_s &= \sum_n \binom{2^s}{n} z^n = (1+z)^{2^s} \end{aligned} \qquad (A.1.6)$$

and iterated

$$B_s = G_s + 2 F_{s-1} B_{s-1}$$

$$= G_s + 2 F_{s-1} G_{s-1} + 2^2 F_{s-1} F_{s-2} B_{s-2}$$

$$= \sum_{0 \le j < s} 2^j F_{s-1} F_{s-2} \cdots F_{s-j} G_{s-j}$$

$$= \sum_{0 \le j < s} 2^j (1+z)^{2^s - 2^{s-j}} [(1+z)^{2^{s-j}} - 1 - 2^{s-j} z] \quad ,$$

yielding

$$A_2(s,n) = 2^s - 1 - \frac{1}{\binom{2^s}{n}} \sum_{0 < j < s} \left[2^j \binom{2^s - 2^{s-j}}{n} + 2^s \binom{2^s - 2^{s-j}}{n-1} \right] . \quad (A.1.7)$$

A.2. Bounds and Asymptotic Estimates for $A_2(s,n)$.

Equation (A.1.7) is of no help in estimating the number of nodes. So, first we will bound $A_2(s,n)$.

Lemma A.2.1. $A_2(s+1,n) > A_2(s,n)$.

Proof. Given a set P in $U_{s+1} = [0,2^{s+1})$, $|P| = n$, define its prefix as the set in $U_s = [0,2^s)$:

$$\text{pref}(P) = \{\lfloor x/2 \rfloor \mid x \in P\} \quad . \quad (A.2.1)$$

Now, the set P can be uniquely generated from $\text{pref}(P)$ by

(i) appending both a "0" and a "1" to the $(|P| - |\text{pref}(P)|)$ elements of $\text{pref}(P)$ such that

$$x_1 \in \text{pref}(P) \quad \text{and} \quad 2x_1, 2x_1+1 \in P \quad (A.2.2)$$

(ii) appending either a "0" or a "1" to the rest of the elements of $\text{pref}(P)$, according to the corresponding element in P .

If we now look at the trie for P, Figure A.2.1, we can see that it has the same node structure as $pref(P)$ with the addition (for each of the elements x_1 defined in (A.2.2) above) of a branch that goes all the way down to the bottom level. Elements like x_2, the rest of $pref(P)$, do not add any new nodes.

So if we partition U_{s+1} into classes according to the size of the prefixes, with

$$p(s+1,n,k) = \text{probability of } (|P| - |pref(P)| = k) \tag{A.2.3}$$

we can express

$$A_2(s+1,n) = \sum_k p(s+1,n,k)[A_2(s,n-k) + k\ Jump(s,n-k)] \tag{A.2.4}$$

where $Jump(s,n-k)$ = average distance between a terminal node and the bottom level s for a trie of $(n-k)$ elements in $U_s = [0,2^s)$ (depicted as branch "J" in Figure A.2.1).

In order to use Equation (A.2.4) we need the following:

Claim A.2.2. $A_2(s,n-1) + Jump(s,n-1) > A_2(s,n)$.

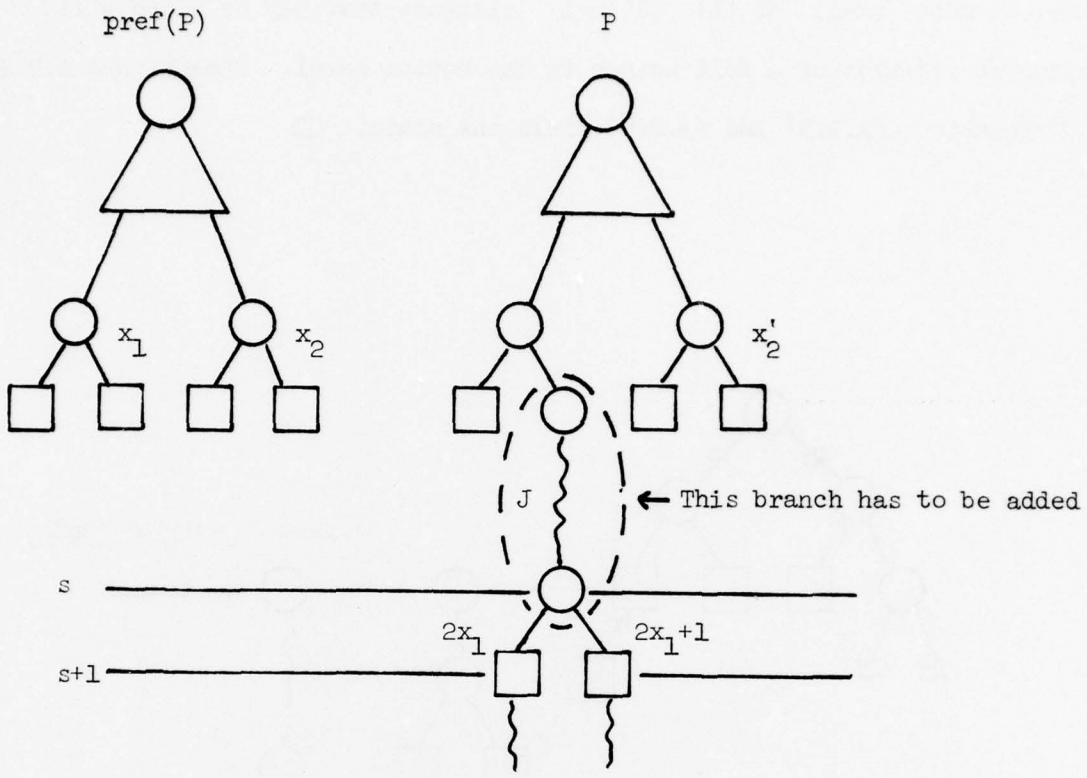
Proof. We can generate $(n-1)$ copies of each set P, $(P \subseteq U_s, |P| = n)$ by taking each one of the elements x in P and adding it to the complementary set $P - \{x\}$.

Thus

$$A_2(s,n) = A_2(s,n-1) + a(s,n-1) \tag{A.2.5}$$

where $a(s,n-1)$ is the average increment when adding a new element

and certainly



(i) x_1 becomes $\{2x_1 , 2x_1+1\}$

(ii) x_2 becomes $x'_2 = 2x'_2 + \alpha$ ($\alpha = 0$ or 1)

Figure A.2.1. Generation of the set P in U_{s+1} .

$$a(s,n-1) < \text{Jump}(s,n-1) \ , \qquad \text{(A.2.6)}$$

since at most $(n-1)$ of the (2^s-n+1) elements that may be added will force the addition of a full branch to the bottom level. (See Figure A.2.2.)

Equations (A.2.5) and (A.2.6) yield the claim. □

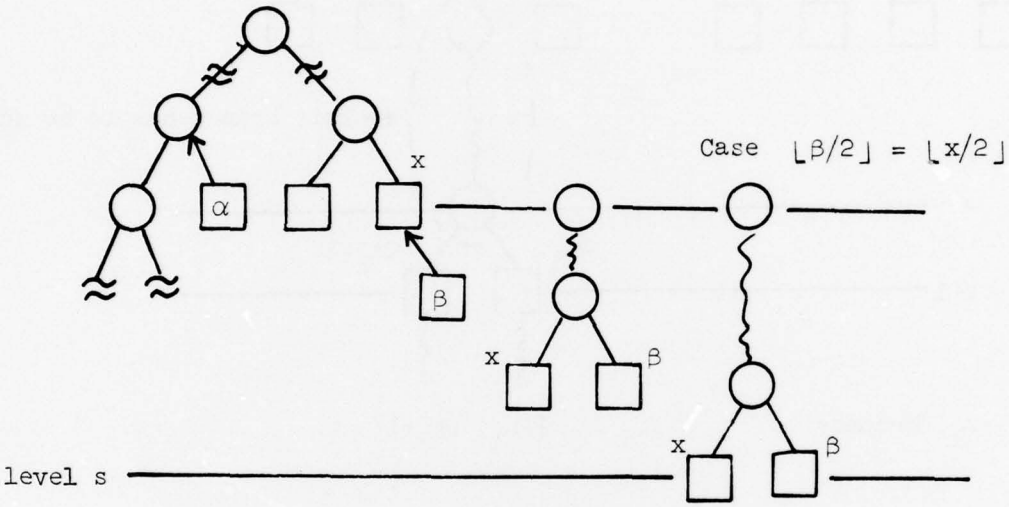


Figure A.2.2. Adding an element to a set of size $(n-1)$.

A new element like α does not increase the node count at all.

A new element like β will produce a branch all the way to the bottom (extreme right) only if $\lfloor \beta/2 \rfloor = \lfloor x/2 \rfloor$.

Going back to our lemma, it is easy to see that

$$A_2(s,n-k) + k\ \mathrm{Jump}(s,n-k) > A_2(s,n) \qquad (A.2.7)$$

given the Claim and the fact that

$$\mathrm{Jump}(s,p) \geq \mathrm{Jump}(s,p+1) \ ,$$

Equation (A.2.4) becomes

$$A_2(s+1,n) > \sum_k p(s+1,n,k)A_2(s,n) = A_2(s,n) \ , \qquad (A.2.8)$$

since $\sum_k p(s+1,n,k) = 1$. □

Once we know that the average number of nodes grows with the size of the universe, we need to know what happens in the limit, when the universe size goes to infinity.

The probability, $\mathrm{part}(n,p,s)$, of a set P_s , $|P_s| = n$, having exactly p elements less than 2^{s-1} may be expressed as

$$\mathrm{part}(n,p,s) = \frac{\binom{2^{s-1}}{n}\binom{2^{s-1}}{n-p}}{\binom{2^s}{n}} = \binom{n}{p}\frac{\binom{2^s-n}{2^{2-1}-p}}{\binom{2^s}{2^{s-1}}}$$

$$= \binom{n}{p}\ \frac{(s^2-n)!}{(2^s)!}\ \frac{(2^{s-1})!}{(2^{s-1}-p)!}\ \frac{(2^{s-1})!}{(2^{s-1}-n+p)!} \qquad (A.2.9)$$

which has the limiting distribution

$$\lim_{s\to\infty} \mathrm{part}(n,p,s) = \binom{n}{p}\frac{1}{2^{sn}} \cdot (2^{s-1})^p \cdot (2^{s-1})^{n-p} = \frac{\binom{n}{p}}{2^n} \ . \qquad (A.2.10)$$

And that happens to be the same probability distribution of the infinite universe $U = [0,1)$ presented in the Introduction and extensively covered in [Knuth 73, Sections 5.2.2 and 6.3].

Hence

Lemma A.2.3. $\lim_{s\to\infty} A_2(s,n) = A(n)$, where $A(n)$ is the average number of nodes of a trie representing a random set of n real numbers in $U = [0,1)$. □

We move on to study $A(n)$ and related topics.

A.3 The Universe of Real Numbers.

This section simply recapitulates some useful results presented in [Knuth 73, Section 5.2.2].

The family of equations

$$\begin{cases} x_n = a_n + 2 \sum_{k\geq 2} \dfrac{\binom{n}{k}}{2^n} x_k & (n \geq 2) \\ x_0 = x_1 = 0 \end{cases} \tag{A.3.1}$$

has solutions

$$x_n = \sum_{k\geq 2} \binom{n}{k} (-1)^k \frac{2^{k-1}\,\hat{a}_k}{2^{k-1}-1} = a_n + \sum_{k\geq 2} \binom{n}{k} (-1)^k \frac{\hat{a}_k}{2^{k-1}-1} \tag{A.3.2}$$

$$\text{where} \quad \hat{a}_n = \sum_k \binom{n}{k} (-1)^k a_k$$

denotes the binomial transform of the sequence $\langle a_k \rangle$. Some of the most popular ones are

$$\begin{aligned} \hat{1} &= \delta_{n0} & \hat{a^n} &= (1-a)^n \\ \hat{n} &= -\delta_{n1} & \widehat{\binom{n}{m}a^n} &= \binom{n}{m}(-a)^m(1-a)^{n-m} \\ \widehat{\binom{n}{m}} &= (-1)^n \delta_{nm} \ . \end{aligned} \tag{A.3.3}$$

The equations above define many properties of tries representing sets in $U = [0,1)$. In particular the number of nodes

$$A(n) = 1 - \delta_{n0} - \delta_{n1} + 2 \sum_{k \geq 2} \binom{n}{k} \cdot 2^{-n} \cdot A(k) \qquad (A.3.4)$$

has

$$\hat{a}_n = \delta_{n0} - 1 + n$$

and thus

$$A(n) = 1 - \delta_{n0} - \delta_{n1} + \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{(\delta_{k0} - 1 + k)}{2^{k-1} - 1}$$

$$= 1 - \sum_{k \geq 2} \binom{n}{k} \frac{(-1)^k}{2^{k-1} - 1} + \sum_{k \geq 2} \binom{n}{k} \frac{(-1)^k k}{2^{k-1} - 1}$$

$$= 1 - U_n + V_n \quad . \qquad (A.3.5)$$

The functions U_n and V_n presented above have the refreshing property of possessing asymptotic expansions:

$$U_n = n \lg n + n \left(\frac{\gamma - 1}{\ln 2} - \frac{1}{2} + f_{-1}(n) \right) + O(1)$$

$$V_n = n \lg n + n \left(\frac{\gamma}{\ln 2} - \frac{1}{2} - f_0(n-1) \right) + O(1) \qquad (A.3.6)$$

The functions $f_{-1}(n)$ and $f_0(n)$ (defined in the answer to Exercise 5.2.2-46 [Knuth 73]) are periodic functions of $\lg n$, and are extremely small:

$$f_{-1}(n) < 2 \times 10^{-7}$$

$$\text{and } f_0(n) < 2 \times 10^{-6} \quad . \qquad (A.3.7)$$

Furthermore both functions have "average value" zero in the following sense: let f be either f_{-1} or f_0, then it turns out that $f(n) = f(\lg n \bmod 1)$ and for random n, $\lg n \bmod 1$ is uniformly distributed (see [Knuth 69; Section 4.2.4]).

Since almost all the quantities that we will be dealing with have asymptotic terms of the form $n \cdot f(n)$, it is very difficult to give theoretically valid asymptotic estimates, though for all practical purposes the $n \cdot f(n)$ terms are negligible.

In any case, and for precision's sake, the concerned reader should regard many of the expressions below (and their counterparts in Chapter 2) simply as <u>good estimates</u> of the leading term of the corresponding asymptotic expansions.

Throughout the rest of the appendix we shall use the catchall term $O(\text{small})$ to denote terms of the form $O(n \cdot f(n))$.

So Equation (A.3.5) becomes

$$A(n) = \frac{n}{\ln 2} + O(\text{small}) + O(1) \quad . \tag{A.3.8}$$

Given the discussion above we shall assume

$$A(n) = \frac{n}{\ln 2} \quad . \tag{A.3.9}$$

A.4 <u>Counting Nodes with Two Nonterminal Siblings</u>.

This quantity, y_n, is used to bound $P_4(s,n)$ in Chapter 2 (Equation 2.1.13)).

A node with n descendants has two nonterminal subtries whenever none of its subtries has size either 0 or 1.

So the quantity a_n in Equation (A.3.1) becomes

$$a_n = 1 - 2(\text{Prob[subtrie of size 0]} + \text{Prob[subtrie of size 1]})$$

$$+ \text{[corrections for n = 0,1,2]}$$

$$= 1 - 2\left(\frac{1}{2^n} + \frac{n}{2^n}\right) + \frac{1}{2}\delta_{n2} + \delta_{n1} + \delta_{n0} \qquad \text{(A.4.1)}$$

that transforms into

$$\hat{a}_n = \delta_{n0} - \frac{2}{2^n} - 2n\left(-\frac{1}{2}\right)\left(\frac{1}{2}\right)^{n-1} + \frac{1}{2}\binom{n}{2} - n + 1$$

$$= \delta_{n0} - 2^{1-n}(2^{n-1} - 1)(n-1) + \frac{1}{2}\binom{n}{2} \quad . \qquad \text{(A.4.2)}$$

So the solution is

$$y_n = a_n - \sum_{k\geq 2}\binom{n}{k}(-1)^k \frac{2^{1-k}(2^{k-1}-1)(k-1)}{2^{k-1}-1}$$

$$+ \sum_{k\geq 2}\binom{n}{k}\frac{1}{2}\binom{k}{2}\frac{(-1)^k}{2^{k-1}-1}$$

$$= a_n + \Sigma_1 + \Sigma_2 \qquad \text{(A.4.3)}$$

where

$$\Sigma_1 = -\sum_{k\geq 2}\binom{n}{k}(-1)^k 2^{1-k}(k-1)$$

$$= \sum_{k\geq 2}\binom{n}{k}k\left(-\frac{1}{2}\right)^{k-1} + 2\sum_{k\geq 2}\binom{n}{k}\left(-\frac{1}{2}\right)^k$$

$$= n\left(\frac{1}{2}\right)^{n-1} - n + 2\left(\frac{1}{2}n - 1 + \frac{1}{2}n\right)$$

$$= n\,2^{1-n} + 2^{1-n} - 2 \qquad \text{(A.4.4)}$$

The second sum Σ_2 is studied in [Knuth 73, Exercise 6.3-19].

$$\Sigma_2 = \frac{1}{2} K(n,2,2) = \frac{1}{4} \frac{n}{\ln 2} + O(\text{small}) + O(1) \quad . \tag{A.4.5}$$

Finally

$$y_n = 1 - 2^{1-n} - n\,2^{1-n} + \frac{1}{2}\delta_{n2} + \delta_{n1} + \delta_{n0} + n\,2^{1-n} + 2^{1-n} - 2$$

$$+ \frac{1}{4} \frac{n}{\ln 2} + O(\text{small}) + O(1)$$

$$= \frac{1}{4} \frac{n}{\ln 2} + O(\text{small}) + O(1) \tag{A.4.6}$$

and we shall adopt

$$y_n = \frac{1}{4} \frac{n}{\ln 2} \approx 0.36\,n \quad . \tag{A.4.7}$$

A.5 Space Requirements for Pointers.

Equation (2.1.19) defines the space x_n needed for pointers under representation 5. But the value of $a_n = (1 - \delta_{n0} - \delta_{n1})(2+\varepsilon) \lg n$ does not allow for an easy solution. So we will take an easy way out: approximate a_n as

$$a_n = (1 - \delta_{n0} - \delta_{n1})(2+\varepsilon)\left[\frac{H_n}{\ln 2} - \frac{\gamma}{\ln 2} \right] \tag{A.5.1}$$

so that

$$p_n = (2+\varepsilon)(x_n - z_n) \tag{A.5.2}$$

where z_n is trivially

$$z_n = \frac{\gamma}{\ln 2} A(n) = \frac{\gamma}{(\ln 2)^2}\, n \tag{A.5.3}$$

and

$$x_n = (1 - \delta_{n0} - \delta_{n1}) \frac{H_n}{\ln 2} + 2 \sum_{k \geq 2} \frac{\binom{n}{k}}{2^n} x_k \quad . \tag{A.5.4}$$

Given that

$$\sum_{k \geq 1} \binom{n}{k} (-1)^k H_k = -\frac{1}{n}$$

we obtain

$$\hat{a}_n = \frac{1}{\ln 2} \left[-\binom{n}{1} (-1) H_1 + \left(-\frac{1}{n} \right) \right] = \frac{1}{\ln 2} \left(n - \frac{1}{n} \right) . \tag{A.5.5}$$

And thus

$$(\ln 2) x_n = (1 - \delta_{n0} - \delta_{n1}) H_n + \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{k - \frac{1}{k}}{2^{k-1} - 1}$$

$$= (1 - \delta_{n0} - \delta_{n1}) H_n + V_n - \Sigma_n \quad , \tag{A.5.6}$$

where

$$\Sigma_n = \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{\frac{1}{k}}{2^{k-1} - 1}$$

$$= \sum_{k \geq 2} \binom{n-1}{k} \frac{1}{k} \frac{(-1)^k}{2^{k-1} - 1} + \frac{1}{n} \sum_{k \geq 2} \binom{n}{k} \frac{(-1)^k}{2^{k-1} - 1}$$

$$= \Sigma_{n-1} + \frac{U_n}{n} \quad . \tag{A.5.7}$$

So

$$\Sigma_n = \sum_{\ell \leq n} \frac{U_\ell}{\ell} = \sum_{\ell \leq n} \lg \ell + n \left(\frac{\gamma - 1}{\ln 2} - \frac{1}{2} \right) + O(\text{small}) + O(1)$$

$$= \lg(n!) + n \left(\frac{\gamma - 1}{\ln 2} - \frac{1}{2} \right) + O(\text{small}) + O(1) \quad . \tag{A.5.8}$$

Replacing into (A.5.6),

$$(\ln 2)x_n = (1-\delta_{n0}-\delta_{n1})H_n + n \lg n + n\left(\frac{\gamma}{\ln 2} - \frac{1}{2}\right)$$

$$- n \lg n - \frac{1}{2}\lg n + \frac{n}{\ln 2} - n\left(\frac{\gamma-1}{\ln 2} - \frac{1}{2}\right) + O(\text{small}) + O(1)$$

$$= (1-\delta_{n0}-\delta_{n1})H_n + 2\frac{n}{\ln 2} + O(\text{small}) + O(1) \quad . \qquad (A.5.9)$$

Yielding

$$x_n = \frac{2}{(\ln 2)^2} n + O(\ln n + \text{small}) \quad . \qquad (A.5.10)$$

And finally we will adopt

$$p_n = (2+\varepsilon)\left[\frac{2}{(\ln 2)^2} n - \frac{\gamma}{(\ln 2)^2} n\right] = (2+\varepsilon)\cdot 2.96n \; . \qquad (A.5.11)$$

The above result is interesting: if pointers are stored as offsets, and encoded according to representation 5, only a constant amount of space (per element in the set) is needed.

A.6 More About the Relation Between the Finite and Infinite Universe.

We would like now to answer the question: how fast does $A_2(s,n)$ approach $A(n)$?

Although at first glance, Equations (A.1.7) and (A.3.5) do not appear to be similar, we can demonstrate that they actually have the same asymptotic behavior for most practical values of n .

Let us first obtain a lower bound for $A_2(s,n)$, transforming Equation (A.3.5)

$$A_2(s,n) = 2^s - 1 - \frac{1}{\binom{2^s}{n}} \sum_{0<j<s} \left[2^j \binom{2^s - 2^{s-j}}{n} + 2^s \binom{2^s - 2^{s-j}}{n-1} \right]$$

$$= \sum_{0<j<s} 2^j + 1 - \sum_{0<j<s} 2^j \frac{\binom{2^s - 2^{s-j}}{n}}{\binom{2^s}{n}}$$

$$- \sum_{0<j<s} 2^s \frac{\binom{2^s - 2^{s-j}}{n-1}}{\binom{2^s}{n}} \quad . \qquad \text{(A.6.1)}$$

We can bound the second sum since

$$\frac{\binom{\alpha-\beta}{n}}{\binom{\alpha}{n}} = \prod_{0 \le k < n-1} \left(1 - \frac{\beta}{\alpha-k} \right) < \left(1 - \frac{\beta}{\alpha} \right)^n \qquad \text{(A.6.2)}$$

and thus

$$\sum_{0<j<s} 2^j \frac{\binom{2^s - 2^{s-j}}{n}}{\binom{2^s}{n}} < \sum_{0<j<s} 2^j (1 - 2^{-j})^n \quad . \qquad \text{(A.6.3)}$$

For the third sum in (A.6.1) notice that

$$\frac{\binom{\alpha-\beta}{n-1}}{\binom{\alpha}{n}} = \frac{n}{\alpha-n+1} \prod_{0 \le k < n-1} \left(1 - \frac{\beta}{\alpha-k} \right) < \frac{n}{\alpha-n+1} \left(1 - \frac{\beta}{\alpha} \right)^{n-1} \qquad \text{(A.6.4)}$$

thus

$$\sum_{0<j<s} 2^s \frac{\binom{2^s - 2^{s-j}}{n-1}}{\binom{2^s}{n}} < 2^s \frac{n}{2^s-n+1} \sum_{0<j<s} (1-2^{-j})^{n-1}$$

$$= n\left(1 + \frac{n-1}{2^s-n+1}\right) \sum_{0<j<s} (1-2^{-j})^{n-1}$$

$$< n \sum_{0<j<s} (1-2^{-j})^{n-1} + \frac{n^2}{2^s-n} \cdot s \quad . \qquad (A.6.5)$$

Finally therefore

$$A_2(s,n) > \sum_{0<j<s} [2^j - 2^j(1-2^{-j})^n - n(1-2^{-j})^{n-1}] - \frac{n^2 \cdot s}{2^s-n} + 1 \quad . \qquad (A.6.6)$$

Now we will transform $A(n)$ as given by Equation (A.3.5)

$$A(n) = 1 + \sum_{k\geq 2} \binom{n}{k} (-1)^k \cdot \frac{k-1}{2^{k-1}-1}$$

$$= 1 + \sum_{k\geq 2} \binom{n}{k} (-1)^k (k-1) \cdot \sum_{0<j} \left(\frac{1}{2^{k-1}}\right)^j$$

$$= 1 + \sum_{0<j} \sum_{k\geq 2} \left[-n\binom{n-1}{k-1}(-1)^{k-1} - \binom{n}{k}(-1)^k \right](2^{-j})^{k-1}$$

$$= 1 - \sum_{0<j} \left[n \sum_{k\geq 1} \binom{n-1}{k}(-2^{-j})^k + 2^j \sum_{k\geq 2} \binom{n}{k}(-2^{-j})^k \right]$$

$$= 1 - \sum_{0<j} [n[(1-2^{-j})^{n-1} - 1] + 2^j[(1-2^{-j})^n - 1 + n2^{-j}]]$$

$$= 1 + \sum_{0<j} [2^j - 2^j(1-2^{-j})^n - n(1-2^{-j})^{n-1}] \quad . \qquad (A.6.7)$$

Comparing Equations (A.6.6) and (A.6.7), and recalling that $A(n) > A_2(s,n)$, it is easy to see that

$$A(n) \;>\; A_2(s,n) \;>\; A(n) - \sum_{s<j} [2^j - 2^j(1-2^{-j})^n - n(1-2^{-j})^{n-1}] - \frac{n^2 s}{2^s - n} \quad . \tag{A.6.8}$$

And we can bound the sum since

$$\begin{aligned}
&\sum_{s<j} [2^j - 2^j(1-2^{-j})^n - n(1-2^{-j})^{n-1}] \\
&\quad = \sum_{k\geq 2} \binom{n}{k} (-1)^k (k-1) \sum_{s<j} \left(\frac{1}{2^{k-1}}\right)^j \\
&\quad < \sum_{k\geq 2} \binom{n}{k} k \; 2 \cdot (2^{-(s+1)})^{k-1} \\
&\quad = 2n \sum_{k\geq 1} \binom{n-1}{k} [2^{-(s+1)}]^k \\
&\quad < 2n \sum_{k\geq 1} \frac{(n-1)^k}{k!} [2^{-(s+1)}]^k \\
&\quad = 2n \cdot \left[\exp\left(\frac{n-1}{2^{s+1}}\right) - 1 \right] = O\left(\frac{n^2}{2^s}\right) \quad .
\end{aligned} \tag{A.6.9}$$

Thus

$$A(n) \;>\; A_2(s,n) \;>\; A(n) - O\left(\frac{n^2 s}{2^s - n}\right) \quad . \tag{A.6.10}$$

And since $A(n) \cong \frac{n}{\ln 2}$ we may conclude that

$$A_2(s,n) = A(n)\left(1 - O\left(\frac{ns}{2^s - n}\right)\right) \quad . \tag{A.6.11}$$

References

[Bloom 70] Burton H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications ACM 13, 7 (July 1970), 422-426.

[Bobrow 75] Daniel G. Bobrow, "A note on hash linking," Communications ACM 18, 7 (July 1975), 413-415.

[Brown and Tarjan 77] M. R. Brown and R. E. Tarjan, "A representation for linear lists with movable fingers," Proc. Tenth Annual ACM Symp. on Theory of Computing, to appear.

[Carter and Wegman 77] J. Lawrence Carter and Mark N. Wegman, "Universal classes of hash functions," IBM Research Report RC 6687, August 1977, 13 pp.

[Coffman and Eve 70] E. G. Coffman Jr., and J. Eve, "File structures using hashing functions," Communications ACM 13, 7 (July 1970), 427-432, 436.

[Diaconis and Freeman 78] Persi Diaconis and David Freedman, "The distribution of the mode of an empirical histogram," Technical Report No. 105, January 1978, Department of Statistics, Stanford University.

[Even and Rodeh 78] S. Even and M. Rodeh, "Economical encoding of commas between strings," Communications ACM 21, 4 (April 1978), 315-317.

[Floyd 77] Robert W. Floyd, personal communication.

[Greniewski and Turski 63] M. Greniewski and W. Turski, "The external language KLIPA for the URAL-2 digital computer," Communications ACM 6, 6 (June 1963), 322-324.

[Knuth 68] Donald E. Knuth, The Art of Computer Programming, vol. 1, Fundamental Algorithms (Addison-Wesley, 1968).

[Knuth 69] Donald E. Knuth, The Art of Computer Programming, vol. 2, Seminumerical Algorithms (Addison-Wesley, 1969).

[Knuth 73] Donald E. Knuth, The Art of Computer Programming, vol. 3, Sorting and Searching (Addison-Wesley, 1973).

[Maly 76] Kurt Maly, "Compressed tries," Communications ACM 19, 7 (July 1976), 409-415.

[Morrison 68] Donald R. Morrison, "PATRICIA - Practical Algorithm To Retrieve Information Coded In Alphanumeric," Journal ACM 15 (1968), 514-534.

[Nicklas and Schlageter 77] B. M. Nicklas and G. Schlageter, "Index structuring in inverted data bases by tries," The Computer Journal 20, 4 (November 1977), 321-324.

[Sprugnoli 77] Renzo Sprugnoli, "Perfect hashing functions: a single probe retrieving method for static sets," Communications ACM 20, 11 (November 1977), 841-850.